Comprando esta edição tenha acesso na WEB a:
+370 vídeo-aulas +150 artigos e 6 cursos online

www.clubedelphi.net Feita para desenvolvedores

ClubeDelphi

Aniversário 7 anos!

Ano 7 · Edição 84 · R$9,90 DevMedia GROUP

ISSN 1517990-7

Deploy Automático

- Minimize o trabalho e esforço de distribuição de suas aplicações Desktop.
- Crie sistemas que são atualizados automaticamente.
- Veja como criar "Service Packs" para atualizar o esquema do banco de dados do cliente.

Na Web:

Confira no portal **ClubeDelphi PLUS** **8 vídeo-aulas** sobre as novidades do

Delphi 2007 for Win32 e Delphi for PHP

- Instalação do Delphi 2007
- Delphi 2007 – Principais Novidades
- Delphi 2007 – Novos Componentes
- Delphi 2007 – Trabalhando com Ajax e Web
- Delphi for PHP – Introdução
- Delphi for PHP – Acessando o MySQL
- Delphi for PHP – Criando um site com BD Parte I
- Delphi for PHP – Criando um site com BD Parte II

Mini-curso
Crie uma aplicação completa usando UML, ASP.NET, POO e ECO – Parte 2

dbExpress
Valide dados de forma eficiente em suas aplicações client/server com Firebird

ASP.NET
DataGrid – Criando colunas dinamicamente e com agrupamento

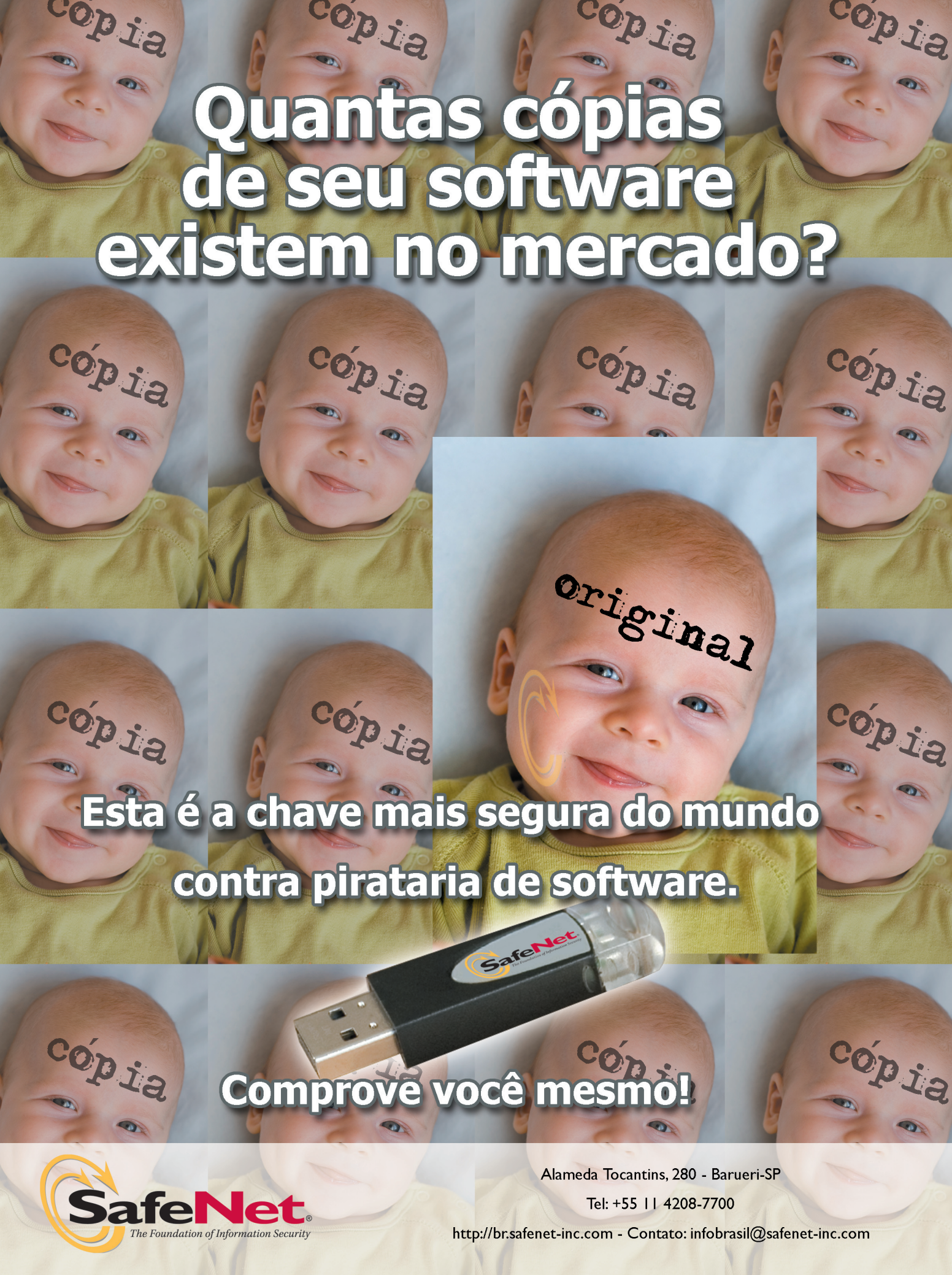
Delphi 2007 for Win32
Tudo sobre a nova versão do Delphi

Gerência de Requisitos
Use o Delphi 2006 com CaliberRM

Sistemas EDI
Faça sua aplicação se comunicar e trocar dados com sistemas de outras empresas

Coluna Ask the Expert

Quantas cópias de seu software existem no mercado?



Esta é a chave mais segura do mundo contra pirataria de software.

Comprove você mesmo!

SafeNet®
The Foundation of Information Security

Alameda Tocantins, 280 - Barueri-SP

Tel: +55 11 4208-7700

http://br.safenet-inc.com - Contato: infobrasil@safenet-inc.com

ClubeDelphi

Ano 7 - 83ª Edição - 2007 - ISSN 1517990-7 **Impresso no Brasil**

Corpo Editorial

Diretor Editorial e Administrativo
Gladstone Matos
gladstone@neoficio.com.br

Designer
Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa
Antonio Xavier
jaxs_design@yahoo.com.br

Articulistas desta edição
Paulo Roberto Quicoli, Fernando Sarturi Prass, Fabrício Desbessel, Frank Ingermann, Paul Vinkenoog, Adriano Santos, Murilo Costa Monteiro Filho, Alberto Germano Sevarolli Trevezani e Marco Antônio Pereira Araújo

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Luciano Pimenta
lucianopimenta@clubedelphi.net

Revisão
Fabio Correa
fabiocorrea@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:
Kaline Dolabella
publicidade@devmedia.com.br

Atendimento ao Leitor
A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Thiago Andrade – Atendimento ao Leitor
www.devmedia.com.br/central/default.asp
(21) 2220-5375

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2220-5375

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Jeff Wendell
jeff@devmedia.com.br

ClubeDelphi PLUS

DevMedia GROUP

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS. Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

Apoio

ASSESPRO PARANÁ

ÍNDICE

06 - Sistemas auto atualizáveis
Marcelo Rodrigo Lopes

10 - Atualização de estruturas de dados
Adriano Santos

16 - Delphi 2007 for Win32
Andreano Lanusse

22 - Desenvolvendo uma Loja Virtual - ASP.NET, UML e ECO – Parte 2
Paulo Roberto Quicoli

28 - Validações em duas camadas
Fabrício Desbessel

34 - DataGrid - Criando colunas dinamicamente
Marcos Santos e Alexandre Santos

40 - Análise de Requisitos com CaliberRM
Felipe La Rocca Teixeira e Marco Antônio P. Araújo

46 - Troca eletrônica de dados por EDI
Adriano Santos

EDITORIAL

O Deploy é uma importante fase do ciclo de vida de uma aplicação. Sistemas construídos em Delphi são distribuídos de forma simplificada, normalmente copiamos o executável e algumas DLLs e BPLs para um diretório compartilhado no servidor. O usuário roda então o aplicativo a partir desse local. Nesta edição, o Marcelo Lopes apresenta uma abordagem mais interessante para facilitar essa distribuição, a criação de sistemas que são atualizados automaticamente. Isso facilita e minimza o esforço de distribuição de suas aplicações desktop, tendo um menor impacto na soliticação de mudança por parte do usuário.

Ainda sobre deploy, um procedimento bastante comum é a necessidade de aplicar alguma modificação no esquema do banco de dados. Quem nunca precisou criar um novo campo em uma tabela para satisfazer a necessidade de um cliente? E quando a estrutura do banco de produção difere do banco de desenvolvimento, e surge aquela indesejável mensagem de erro na tela do usuário? Sincronizar as versões do banco e da aplicação é algo complicado. O Adriano Santos nesta edição apresenta uma solução para este problema, onde é possível aplicar uma espécie de "service pack" no banco toda vez que o esquema precisar ser atualizado. Ideal para bancos que sofrem com atualizações constantes na estrutura.

Ainda nesta edição, um especial sobre o Delphi 2007 for Win32, no artigo de Andreano Lanusse. Confira todas as novidades da última versão do Delphi da CodeGear. Não perca também imperdível mini-curso do Paulo Quicoli, sobre o desenvolvimento de uma aplicação Web completa usando POO, UML, ECO e ASP.NET. Confira ainda muito sobre ASP.NET, DataGrid, Delphi e Caliber e transferência eletrônica de dados.

Bom leitura e sucesso com o Delphi.

Guinther Pauli
guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET
Borland Certified: Delphi 6, 7, 2005, 2006, Web, Kylix

Fale com o Editor

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site .NET Magazine, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

Luciano Pimenta - Editor do Site
lucianopimenta@clubedelphi.net

Portal do Assinante

NÃO PERCA

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:
- Mais de 360 Vídeo Aulas!
- 6 cursos online!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 140 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse *www.devmedia.com.br/clubedelphi/portal.asp* e utilize as informações abaixo: **Login: DVM.CD e Senha: A54BR**

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Ask The Expert

Perguntas e Respostas

Pesquisa auto-completar

Como faço pra criar uma pesquisa com auto-completar?

Paulo Hasper

Primeiro insira no formulário um *Edit* e um *ListBox*. Preencha o *ListBox* com alguns nomes de pessoas, logo em seguida crie uma função com o código da **Listagem 1**.

Declare também uma variável global com o nome de "bPausa" do tipo *Boolean*. Então no evento *OnChange* do *Edit* digite o código da **Listagem 2**.

E no evento *OnKeyDown* também do *Edit* digite o código da **Listagem 3**.

Dessa forma enquanto o usuário estiver digitando o nome no *Edit* o mesmo será preenchido com o nome encontrado no *ListBox* (**Figura 1**).

Listagem 1. Código de busca

```
function TForm1.Procura(Txt: string): string;
var
  iA: Integer;
begin
  with ListBox1.Items do
    if TXT <> '' then
      for iA := 0 to Pred(Count) do
        if (lStrCmp(PChar(Copy(Strings[iA], 1,
          Length(TXT))), PChar(TXT)) = 0) or
          (lStrCmpi(PChar(Copy(Strings[iA], 1,
          Length(TXT))), PChar(TXT)) = 0) then
        begin
          Result := Strings[iA];
          Break;
        end;
end;
```

Listagem 2. Evento OnChange do Edit

```
procedure TForm1.Edit1Change(Sender: TObject);
var
  iA, iB: Integer;
begin
  if not bPAUSA and Active then
  begin
    if Procura(Edit1.Text) <> '' then
    begin
      iB := Length(Edit1.Text);
      Edit1.Text := Procura(Edit1.Text);
      Edit1.Selstart := iB;
      Edit1.SelLength := Length(Procura(
        Edit1.Text)) - iB;
    end;
  end;
  bPausa := False;
  for iA := 0 to ListBox1.Items.Count - 1 do
    if ListBox1.Items.Strings[iA] = Edit1.Text then
    begin
      ListBox1.ItemIndex := iA;
      Break;
    end else
      ListBox1.ItemIndex := -1;
end;
```

Listagem 3. Evento OnKeyDown do Edit

```
procedure TForm1.Edit1KeyDown(Sender: TObject;
  var Key: Word; Shift: TShiftState);
begin
  if (KEY = VK_BACK) or (KEY = VK_DELETE) then bPAUSA := True;
  if KEY = 13 then Edit1.SelStart := Length(Edit1.Text);
  if KEY = 27 then Edit1.Clear;
end;
```

Como ler e modificar as configurações regionais?

Como faço pra recuperar e/ou alterar as configurações de data e decimais das Configurações Regionais da máquina que está rodando o meu projeto?

Lucas Oliveira

Para responder a essa pergunta, vamos verificar a unit *SysUtils* do Delphi, que possui algumas variáveis responsáveis por receber e/ou gravar as configurações regionais. Algumas dessas variáveis são:

```
var
  CurrencyString: string;
  CurrencyFormat: Byte;
  ThousandSeparator: Char;
  DecimalSeparator: Char;
```

Pra lermos basta incluir o seguinte código:

```
Edit1.Text := DecimalSeparator;
Edit2.Text := ThousandSeparator;
Edit3.Text := ShortDateFormat;
Edit4.Text := LongDateFormat;
```

Veja na **Figura 2** a apresentação dessas variáveis.

E para gravar, basta passarmos novamente os valores para as variáveis e as configurações regionais estarão alteradas.

```
DecimalSeparator := Edit1.Text;
ThousandSeparator := Edit2.Text;
ShortDateFormat := Edit3.Text;
LongDateFormat := Edit4.Text;
```

Adriano Santos

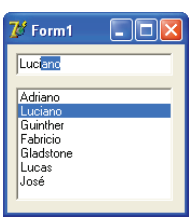


Figura 1. Edit com auto-completar

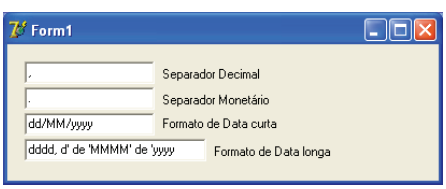


Figura 2. Exibindo as variáveis com os dados das configurações regionais

Novidades do ClubeDelphi PLUS

+360 vídeo aulas!

6 cursos online

www.clubedelphi.net/portal

Caro Leitor,

O portal ClubeDelphi PLUS é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com:

i) *Mais de 360 vídeo aulas;*

ii) *6 cursos online;*

iii) *1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP;*

iv) *Mais de 150 artigos exclusivos (que não foram publicados na revista)!;*

Acesse o portal ClubeDelphi PLUS e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal (para quem compra em bancas, o acesso é válido por 30 dias).

Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou em bancas, utilize o login e senha publicados na página do editorial desta edição.

Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!

Equipe DevMedia

.:: DESTAQUE ::.

Confira no portal ClubeDelphi PLUS 8 vídeo aulas sobre Delphi for PHP e Delphi 2007:

Instalação do Delphi 2007
http://www.devmedia.com.br/articles/viewcomp.asp?comp=5002

Novidades do Delphi 2007 – Parte I
http://www.devmedia.com.br/articles/viewcomp.asp?comp=5079

Novidades do Delphi 2007 – Parte II
http://www.devmedia.com.br/articles/viewcomp.asp?comp=5172

Novidades do Delphi 2007 – Parte III
http://www.devmedia.com.br/articles/viewcomp.asp?comp=5263

Aplicações MySQL com Delphi for PHP – Parte I
http://www.devmedia.com.br/articles/viewcomp.asp?comp=4985

Aplicações MySQL com Delphi for PHP – Parte II
http://www.devmedia.com.br/articles/viewcomp.asp?comp=5056

Aplicações MySQL com Delphi for PHP – Parte III
http://www.devmedia.com.br/articles/viewcomp.asp?comp=5214

Aplicações MySQL com Delphi for PHP – Parte IV
http://www.devmedia.com.br/articles/viewcomp.asp?comp=5238

Últimas Vídeo-Aulas **(Somente para assinantes)**

Sistema completo com Delphi 7, dbExpress e Firebird 2.0
Acompanhe a criação de um sistema completo de gerenciamento de locadora, usando Delphi, dbExpress e Firebird 2.0. Luciano Pimenta, mostrará todos os passos necessários para a criação do projeto em todas as fases de implementação, desde a criação do banco de dados, até a migração para .NET.

Executando métodos de forma assíncrona
Veja nessa vídeo aula de Paulo Quicoli, como executar métodos de forma assíncrona.

Comparando banco de dados com o DBComparer
Veja nessa vídeo aula de Everson Volaco, como comparar estruturas de bancos com o DBComparer.

Criando um sistema de backup do Firebird no Delphi
Veja nessa vídeo aula de Everson Volaco, como criar um sistema de backup do IB/FB no Delphi.

Instalação do Delphi 2007
Veja nessa vídeo aula de Adriano Santos, como baixar e instalar uma versão trial do Delphi 2007.

Novidades do Delphi 2007 - Parte I a III
Veja nessas aulas de Adriano Santos as principais novidades em relação ao novo release do Delphi, o Delphi 2007.

Aplicações MySQL com Delphi for PHP - Parte I a IV
Veja nessas vídeo aulas de Fabricio Desbessel como trabalhar com MySQL no Delphi for PHP.

Sistemas auto atualizáveis

MARCELO RODRIGO LOPES
(marolo@pop.com.br)
é desenvolvedor. Possui 8 anos de experiência no desenvolvimento de sistemas comerciais e ERP.

Quando vamos implantar um sistema em muitas estações umas das primeiras preocupações é como atualizar todas as estações com novas versões? Qual a melhor solução? Uma solução muito utilizada é a criação de um atalho na estação, apontando para um caminho de rede, é uma boa solução?

Sim, porém para determinados sistemas é inviável, imagine um sistema baseado em DLLs, o mesmo poderia não funcionar corretamente com a solução do atalho. Então como proceder? Uma ótima solução seria desenvolver um "aplicativo" que atualize seu sistema automaticamente.

Mostrarei neste artigo como desenvolver esse aplicativo, o mesmo será constituído em duas partes, o *Administrador*, responsável por indicar os arquivos que fazem partes do sistema e que devem ser atualizados, e o *Atualizador*, aplicativo instalado nas estações, responsável por atualizar o sistema toda vez que for inicializado.

Preparação do banco de dados

Antes de começarmos a criar os aplicativos, precisamos fazer uma alteração no banco de dados para guardar os arquivos que compõem o sistema. Mostrarei o exemplo em Firebird, por ser um banco de dados gratuito e de fácil conexão, através da tecnologia dbExpress. Fique a vontade de usar outro banco, pois as diferenças são mínimas. Na **Listagem 1** temos o código para criação da tabela.

Precisamos entender a necessidade dos campos da tabela FILES_SYS:

- *HANDLE*: Chave primária da tabela;
- *FILE_NAME*: Nome do arquivo incluído no sistema;
- *FILE_*: Arquivo propriamente dito. Lembrando que o tipo BLOB possui a capacidade de armazenar arquivos binários, por isso ele está sendo utilizado, vamos guardar os arquivos do sistema diretamente no banco de dados (para outro banco de

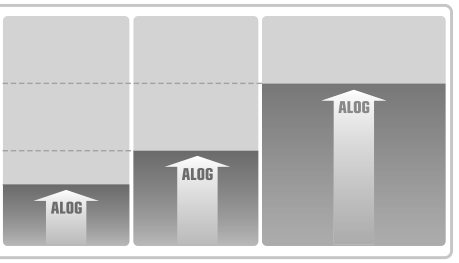
FAÇA COMO A ALOG
FOQUE EM SEU NEGÓCIO

A ALOG é totalmente focada em serviços de Data Center. Dedica-se 24 horas por dia, 7 dias por semana a cuidar do negócio dos nossos 320 clientes corporativos.

ALOG desenvolveu no Brasil o conceito de **HOSTING GERENCIADO**: um serviço de excelência que garantirá a sua empresa estabilidade, agilidade, redundância, e escalabilidade. Para que isso seja verdade os clientes da ALOG contam com **Suporte Incondicional***, um conceito revolucionário em atendimento no Brasil.

ENTENDA O **HOSTING GERENCIADO**

CAMADAS DE NEGÓCIO	DESCRIÇÃO
Processo de Negócio	Atividades cotidianas do cliente
Aplicação do Cliente	Softwares de Gestão (Ex: ERP, CRM, outros)
Ambiente da Aplicação	.NET, JAVA, SQL Server, MySQL, Oracle, Exchange
Sistema Operacional	Windows, Linux, Unix, Solaris
Servidor ou equipamento	Servidores RISC/CISC, Appliances, Roteadores, Switches
Rede	IP, Wan, Rede Local
Infra-estrutura Básica	Energia, HVAC, Segurança - 2N+1



ESTRATÉGIA DO CLIENTE	COLOCATION	HOSTING	HOSTING GERENCIADO

PLANOS DE **HOSTING GERENCIADO**

GERENCIADO 1

- **Pentium D 2.8 GHz**
- 512 RAM DDR2
- 2 HD SATA 80 GB
- 450 GB de Transf. Mensal
- Firewall e Backup

BOOT REMOTO
GERENCIAMENTO**

R$**1.050,00** mensais

INSTALAÇÃO IMEDIATA***

GERENCIADO 2

- **Dual Xeon Dual Core 3.0 GHz**
- 1GB RAM DDR2 533
- 2 HD SATA RAID I 300 GB
- 1.000 GB de Transf. Mensal
- Firewall e Backup

BOOT REMOTO
GERENCIAMENTO**

R$**2.850,00** mensais

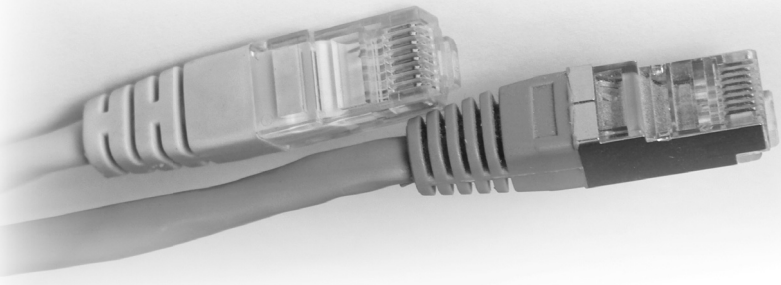
INSTALAÇÃO IMEDIATA***

* Sujeito a condições contratuais ** Podendo escolher: HD, RD e GT *** Sujeito a disponibilidade no estoque

ALOG data centers

Excelência em Hosting
www.alog.com.br :: 0800 282 3330

Hosting Gerenciado I Colocation I Contingência I Email Corporativo I Conectividade I Serviços Profissionais



dados, verifique qual o tipo que armazena arquivos binários);

• *LAST_UPDATE*: Data da última atualização do arquivo.

O aplicativo Manager

O próximo passo é desenvolver o aplicativo *Manager*, responsável pela inserção, alteração e exclusão de arquivos no sistema. Crie uma aplicação e crie um único formulário, faça a conexão com o banco de dados, acessando a tabela FILES_SYS. Em seguida monte uma tela semelhante a **Figura 1**.

Em seguida vamos codificar os botões. Na **Listagem 2**, temos o código do *New File*.

O *New File* inclui na tabela um registro contendo o próprio arquivo pertencente ao sistema, com seu nome e data de criação. Codificaremos agora o *Update*, conforme a **Listagem 3**.

O *Update* substitui o arquivo selecionado pelo novo. Quando você necessita enviar atualizações aos clientes, esse será o caminho utilizado. Codificaremos em seguida o *Delete File*, conforme a **Listagem 4.**

O *Dele File* exclui um arquivo do sistema, caso o mesmo não seja mais necessário. Um último detalhe do sistema será um código no evento *OnDataChange* do *DataSource*, conforme a **Listagem 5.**

Dessa forma, evitamos que *Update* e *Delete File* fiquem habilitados indevidamente.

O aplicativo Atualizador

A função desse aplicativo é verificar os arquivos existentes em FILES_SYS, fazer a comparação com os arquivos existentes na estação e copiar os mesmos que estiverem diferentes (note que estaremos atualizando todos os arquivos **diferentes** e não somente os mais recentes, consideremos que os arquivos válidos são os que existem no servidor).

Esse aplicativo é simples, possui uma conexão com o banco

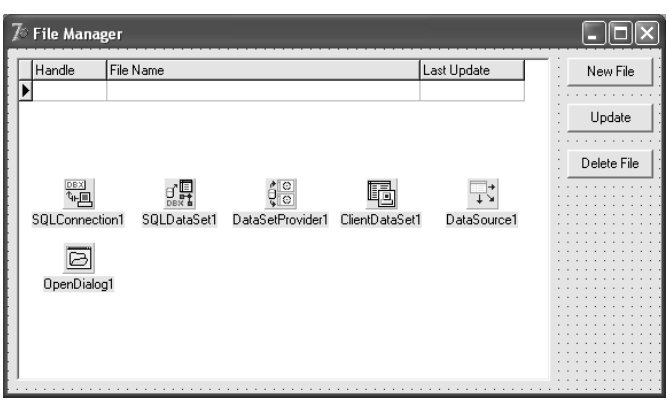


Figura 1. Tela do aplicativo Manager

Listagem 1. Código de criação da tabela

```
CREATE TABLE FILES_SYS (
 HANDLE      INTEGER NOT NULL,
 FILE_NAME   VARCHAR(100) NOT NULL,
 FILE_       BLOB NOT NULL,
 LAST_UPDATE TIMESTAMP NOT NULL,
 PRIMARY KEY (HANDLE))
```

Listagem 2. Código do New File

```
procedure TFMenu.BtnNewFileClick(Sender: TObject);
var
  viHandle: Integer;
begin
  if OpenDialog1.Execute then
  begin
    ClientDataSet1.Last;
    { último handle + 1 }
    viHandle := ClientDataSet1HANDLE.AsInteger + 1;
    { Inserir um novo registro }
    ClientDataSet1.Append;
    ClientDataSet1HANDLE.AsInteger := viHandle;
    ClientDataSet1FILE_NAME.AsString :=
      ExtractFileName(OpenDialog1.FileName);
    ClientDataSet1LAST_UPDATE.AsDateTime :=
      FileDateToDateTime(FileAge(
      OpenDialog1.FileName));
    ClientDataSet1FILE_.LoadFromFile(
      OpenDialog1.FileName);
    ClientDataSet1.Post;
    ClientDataSet1.ApplyUpdates(0);
  end;
end;
```

Listagem 3. Código do Update

```
procedure TFMenu.BtnUpdateClick(Sender: TObject);
begin
  if OpenDialog1.Execute then
  begin
    ClientDataSet1.Edit;
    ClientDataSet1FILE_NAME.AsString :=
      ExtractFileName(OpenDialog1.FileName);
    ClientDataSet1LAST_UPDATE.AsDateTime :=
      FileDateToDateTime(FileAge(
      OpenDialog1.FileName));
    ClientDataSet1FILE_.LoadFromFile(
      OpenDialog1.FileName);
    ClientDataSet1.Post;
    ClientDataSet1.ApplyUpdates(0);
  end;
end;
```

Listagem 4. Código do Delete File

```
procedure TFMenu.BtnDeleteClick(Sender: TObject);
begin
  if MessageDlg('Confirma a exclusão do arquivo "'+
    ClientDataSet1FILE_NAME.AsString+'" ?',
    mtconfirmation, [mbyes,mbno],0) = mryes then
  begin
    ClientDataSet1.Delete;
    ClientDataSet1.ApplyUpdates(0);
  end;
end;
```

Listagem 5. Código do evento OnDataChange

```
procedure TFMenu.DSCDSFilesDataChange(
  Sender: TObject; Field: TField);
begin
  btnUpdate.Enabled := not DataSource1.DataSet.IsEmpty;
  btnDelete.Enabled := not DataSource1.DataSet.IsEmpty;
end;
```

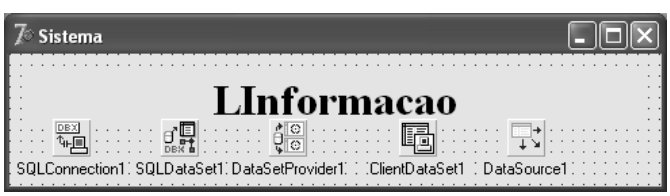


Figura 2. Formulário principal para o aplicativo atualizador

de dados, assim como o *Manager*, e um único código, razoavelmente simples.

Crie uma nova aplicação, faça semelhante a anterior, conectando ao banco e a tabela, em seguida monte uma tela semelhante a **Figura 2**.

No evento *OnShow* do formulário, codificaremos conforme a **Listagem 6**.

Esse código passa por todos os arquivos existentes em FILES_SYS e compara um a um com o arquivo existente localmente, caso o mesmo esteja desatualizado, o aplicativo copiará o mesmo para a estação.

Observe que utilizamos o *FormatDateTime*, para evitar que possíveis diferenças de configurações de data/hora entre as estações faça com que o sistema copie arquivos desnecessários. Pronto, agora basta rodar o aplicativo *Manager*, incluir os arquivos pertencentes ao seu sistema e em seguida, copiar o aplicativo *Atualizador*, nas estações e executar.

O que pode ser feito também é a inclusão de um código após a atualização chamando o executável principal do sistema, dessa forma, o usuário não necessitará executar dois aplicativos toda vez que entrar no sistema.

Listagem 6. OnShow do formulário

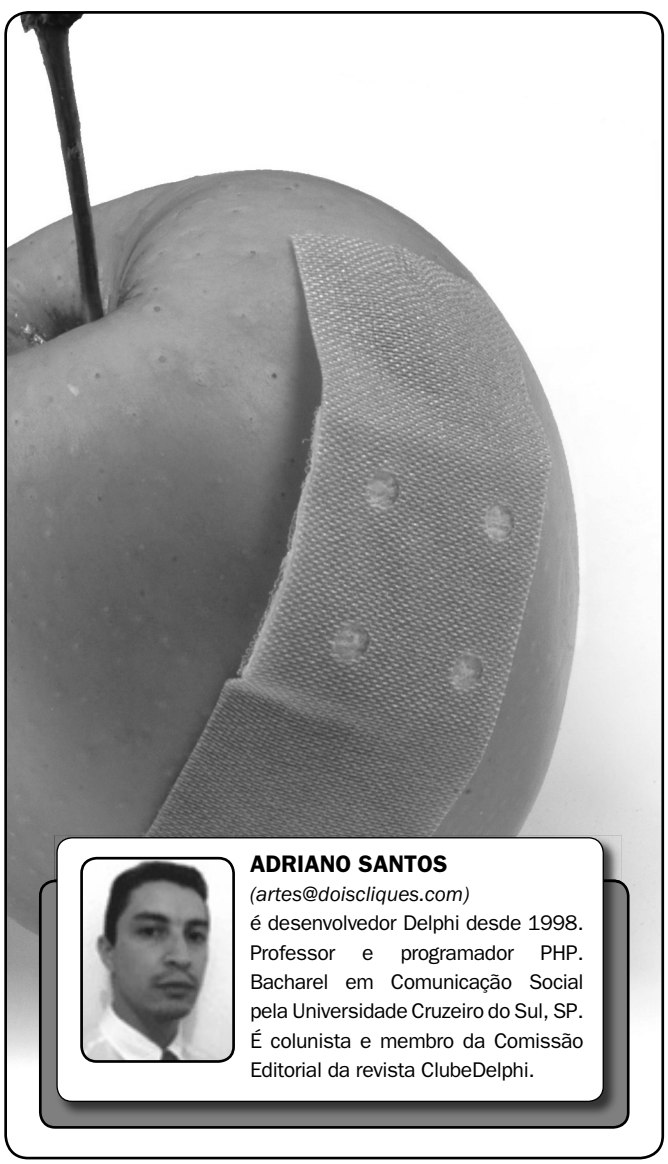
```
procedure TFManu.FormShow(Sender: TObject);
var
  vDirectory: string;
begin
  LInformacao.Caption := 'Atualizando arquivos do sistema';
  vDirectory := ExtractFileDir(Application.ExeName)+'\';
  ClientDataSet1.Open;

  { Primeiro arquivo da lista }
  CDSFiles.First;
  while not(ClientDataSet1.EOF) do
  begin

    { Atualização da tela do sistema }
    Application.ProcessMessages;

    if not FileExists(vDirectory +
      ClientDataSet1FILE_NAME.AsString) then
      ClientDataSet1FILE_.SaveToFile(vDirectory +
        ClientDataSet1FILE_NAME.AsString)

    else
      if FormatDateTime('dd/mm/yyyy hh:mm:ss',
        FileDateToDateTime(FileAge(vDirectory +
        ClientDataSet1FILE_NAME.AsString))) <>
        FormatDateTime('dd/mm/yyyy hh:mm:ss',
        ClientDataSet1LAST_UPDATE.AsDateTime) then
          ClientDataSet1FILE_.SaveToFile(vDirectory +
          ClientDataSet1FILE_NAME.AsString);
    ClientDataSet1.Next;

  end;
  ClientDataSet1.Close;
  SQLConnection1.Close;
  LInformacao.Caption := 'Sistema atualizado';
end;
```

Conclusão

A utilização dessa forma de atualização evita que alguém tenha que sair copiando arquivos de estação por estação quando uma atualização no sistema for realizada, agilizando muito o funcionamento do sistema.

Lembro que essa é uma forma muito proveitosa para sistemas compostos por muitos arquivos (DLL, INI, EXE etc.), para sistema onde um único executável é necessário, essa não seria uma boa prática a ser adotada. ■

Atualização de Estruturas de Dados



ADRIANO SANTOS
(artes@doiscliques.com)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É colunista e membro da Comissão Editorial da revista ClubeDelphi.

Gostaria de iniciar este artigo voltando ao passado quando os SGBD's nem eram tão utilizados no dia a dia e SQL parecia uma linguagem extremamente difícil tanto de entender quanto de usar. Nem sou tão velho assim, mas me lembro como era difícil atualizar o banco de dados de meus clientes feito em Paradox. Era tabela pra cá, telefonema pra lá e muitos, muitos e-mails.

Hoje a situação mudou, aliás, de um bom tempo pra cá, às coisas tem mudado. Porém ainda me perguntam freqüentemente: "*Como atualizar a base de dados de maneira rápida e prática*?". Todos sabemos que desenvolver aplicações comerciais significa que, além de um sistema bem estruturado, uma base de dados confiável, estável e sólida são de suma importância.

Um dos elementos básicos, entre vários, pra ter isso, é trabalhar com um banco compatível com a aplicação. Significa que campos, tabelas e uma série de outros objetos devem existir na base do cliente exatamente igual a estrutura de desenvolvimento.

Ou seja, é no mínimo inadmissível que um campo programado pra receber determinado valor não exista no SGBD do cliente o que acarretaria um *crash* do sistema nas telas que o utilizam. Esse é apenas um dos por menores que podem ocorrer em um sistema incompatível com seu próprio bando de dados.

Esse é o assunto deste artigo. Vamos desenvolver, didaticamente, um atualizador automático usando *patches*.

> **Nota:** *Segundo o Wikipedia (pt.wikipedia.org/wiki/Patch) um patch é um programa de computador do sistema operacional UNIX que aplica as diferenças textuais entre dois programas e, mais freqüentemente, a arquivos de computador contendo essas diferenças, ou arquivos diff.*

Bem, em nosso caso o *patch* será na verdade um arquivo de *script* renomeado de SQL para PATCH, o que não deixa de ser

uma boa idéia, assim diferenciamos o arquivo de outros da pasta de instalação.

A idéia é que o desenvolvedor crie o *patch* da forma que achar mais conveniente. No meu dia a dia gosto de trabalhar com o IBExpert, pois desenvolvo minhas aplicações em Firebird. Rapidamente posso criar um *script* apenas extraindo o metadados do banco de testes e gerar o *script*. Após, envio para o cliente que utiliza alguma ferramenta que faça o processo inverso, como o IBConsole ou IBOConsole.

Essa não é uma boa prática, pois preciso forçar o cliente a ter um desses programas, instalado e configurado pra então executar a tarefas de atualizar o banco. Com nosso atualizador automático, podemos focar apenas em desenvolver o *script* correto para atualização do banco.

Então basta enviar os arquivos por e-mail ou desenvolver alguma ferramenta capaz de fazer o download dos *patches* de um servidor FTP o que tornaria o processo ainda menos dependente do usuário.

A classe TPatche

Para este artigo preparei a criação de uma classe capaz de receber o *patch* e executá-lo. Encapsuladamente conectamos ao servidor usando dbExpress e enviamos todos os comandos necessários para a execução de cada linha de comando encontrada no *script*.

A classe está preparada apenas para trabalhar com banco de dados IB/FB, mas pode ser facilmente portada pra outras bases como Oracle, MySQL entre outras. Outra coisa importante a falar, é que não serão permitidos comandos como criação de *Stored Procedures*, *Triggers*, *Exceptions* etc.

Apenas comandos como *Insert*, *Delete*, *Update*, *Alter Table*, *Create Table* entre outros poderão ser utilizados para execução. Um exemplo de *script* pode ser visto na **Listagem 1**.

Listagem 1. Exemplo de patch de atualização

```
CREATE GENERATOR GEN_VISITANTES_ID;
SET GENERATOR GEN_VISITANTES_ID TO 10000;

CREATE GENERATOR GEN_VISITAS_ID;
SET GENERATOR GEN_VISITAS_ID TO 50000;

CREATE TABLE VISITANTES (
  ID          INTEGER NOT NULL,
  NOME        VARCHAR(5),
  ENDERECO    VARCHAR(100),
  CIDADE      VARCHAR(100),
  ESTADO      VARCHAR(2),
  RG          VARCHAR(15),
  CPF         VARCHAR(15),
  DATA_NASC   TIMESTAMP,
  TELEFONE    VARCHAR(15),
  EMPRESA     VARCHAR(60),
  CONTATO     VARCHAR(30),
  DATA_CAD    TIMESTAMP,
  FOTO        VARCHAR(200));

INSERT INTO VISITANTES
  (ID, NOME, ENDERECO, CIDADE,
  ESTADO, RG, CPF, DATA_NASC, TELEFONE, EMPRESA,
  CONTATO, DATA_CAD, FOTO)
VALUES
  (1, 'ADFDA', NULL, NULL, NULL, '22.222.222-2',
  NULL, NULL, NULL, NULL, NULL, '2007-03-21 00:04:41',
  'C:\Developer Delphi\Articles\My Articles\1-Sends\Webcam\
      Exemplo\222222222.jpg');

ALTER TABLE VISITANTES ADD PRIMARY KEY (ID);
UPDATE CONFIGURACOES SET VERSAO_DO_BANCO = 5104;
```

Preparando a aplicação

Pra entendermos melhor como tudo funcionará nada como desenvolver na prática. Então entre no Delphi e crie um novo projeto clicando em *File>New>Application*. Vamos necessitar desenhar um formulário semelhante à **Figura 1**.

No centro da tela estão dois componentes do tipo *FileListBox* com os nomes *flbPatches* (esquerda) *flbLogs* (direita) que mostrarão os arquivos de *patch* e *log* respectivamente. O restante são elementos comuns, como *Label* e um *ProgressBar*, preenchida a medida que os *patches* vão sendo executados.

> **Nota:** *Enquanto o script estiver sendo executado não é possível atualizar a barra de progresso por se tratar de um comando SQL, portanto recomendo que crie formas de informar ao usuário final que o processo está sendo executado como a exibição de animações e/ou Gifs.*

Criando a classe TPatches

Antes de desenvolver a classe nomeie o formulário para "frmPatche" e salve-o como "uPatche.pas". Logo em seguida salve o projeto como "Patches.dpr". Vamos criar uma nova *unit* onde faremos o desenvolvimento da classe.

Para isso clique em *File>New>Unit*. Salve-a como "cPatche.pas" no diretório do projeto. Vamos declarar o que será necessário para que a classe funcione. Crie a seção *uses* abaixo de *interface* as seguintes units:

```
Db, DbTables, MidasLib, Windows, Forms, Dialogs,
Messages, SysUtils, SqlExpr, Classes, DBXpress;
```

Agora faremos à implementação da estrutura básica da classe, declarando as funções, procedimentos, propriedades e campos necessários. Crie uma nova seção *type* como mostrado na **Listagem 2**.

Figura 1. Exemplo de janela do atualizador

Listagem 2. Declaração da estrutura principal da classe

```
type
  TPatches = class(TObject)
  private
    FPatch: string;
    FConexao: TSqlConnection;
    function ConectarAoBanco: Boolean;
    procedure Desconectar;
    function GetPatch: string;
    procedure SetPatch(const Value: string);
  public
    function Execute: Boolean;
    property Patch: string read GetPatch write SetPatch;
  end;
```

Pressione CTRL+SHIT+C para que o Delphi crie o cabeçalho de todos os procedimentos e funções na área *implementation*. Vamos começar a escrever os códigos de *GetPatch* e *SetPatch*, responsáveis por receber e gravar, respectivamente, o caminho do *patch* na classe.

Em *GetPatch* apenas retornamos o valor da variável *FPatch*, conforme o seguinte código:

```
function TPatches.GetPatch: string;
begin
  Result := FPatch;
end;
```

Em *SetPatch* recebemos o valor de *FPatch* e em seguida verificamos se ele realmente existe, já que nessa propriedade receberemos o caminho completo do arquivo de *script*, conforme o código da **Listagem 3**.

Listagem 3. Implementando o SetPatch

```
procedure TPatches.SetPatch(const Value: string);
begin
  FPatch := Value;
  if not FileExists(FPatch) then
  begin
    MessageBox(0, PChar('O arquivo de patch não '+
      'existe ou o seu caminho está incorreto.' +
      #13#13 + FPatch), 'Erro', MB_ICONERROR);
    Abort;
  end;
end;
```

Codificaremos a *function* e a *procedure* responsável por conectar e desconectar, respectivamente, do banco de dados. São elas: *ConectarAoBanco* e *Desconectar*.

> **Nota:** *Vale lembrar que este exemplo foi concebido visando apenas IB/FB, portanto a função de conexão não foi preparada pra multi bancos. Caso haja interesse em saber como montar essa estrutura, desenvolvi um artigo que encontra-se no portal DevMedia falando sobre o assunto.*

Desconectar é bastante simples. Consiste apenas em verificar se há uma conexão ativa e então desconectá-la, liberando também as variáveis de conexão da memória, como temos na **Listagem 4**.

Listagem 4. Código do método Desconectar

```
procedure TPatches.Desconectar;
begin
  if Assigned(FConexao) and FConexao.Connected then
  begin
    FConexao.Connected := False;
    FConexao.Free;
    FConexao := nil;
  end;
end;
```

Por fim vamos à função de conexão com o banco. Na **Listagem 5** encontramos todo o código da *ConectarAoBanco*. Ela inicializa a variável *FConexao* criando um *SQLConnection* em *run time*. Logo em seguida, ele é inicializado com o valores referentes a nossa conexão, incluindo o caminho do banco de dados.

Listagem 5. Função de conexão com o banco

```
function TPatches.ConectarAoBanco: Boolean;
begin
  try
    if not Assigned(FConexao) then
    begin
      FConexao := TSqlConnection.Create(Application);
      with FConexao do
      begin
        ConnectionName := 'INTERBASE';
        DriverName := 'Interbase';
        GetDriverFunc := 'getSQLDriverINTERBASE';
        LibraryName := 'dbexpint.dll';
        LoginPrompt := False;
        Params.Clear;
        Params.Add('DriverName=Interbase');
        Params.Add(
          'Database=127.0.0.1/3050:C:\ClubeDelphi\'+
          'Patches.fdb');
        Params.Add('RoleName=RoleName');
        Params.Add('User_Name=SYSDBA');
        Params.Add('Password=masterkey');
        Params.Add('ServerCharSet=');
        Params.Add('SQLDialect=1');
        Params.Add('BlobSize=-1');
        Params.Add('CommitRetain=False');
        Params.Add('WaitOnLocks=True');
        Params.Add('ErrorResourceFile=');
        Params.Add('LocaleCode=0000');
        Params.Add(
          'Interbase TransIsolation=ReadCommited');
        Params.Add('Trim Char=False');
        VendorLib := 'gds32.dll';
      end;
    end;
    FConexao.Connected := True;
    Result := True;
  except on E: Exception do
    begin
      MessageDlg('Ocorreu um erro ao conectar-se '+
        'ao banco.' + #13 + E.Message, mtError, [mbOk], 0);
      Result := False;
    end;
  end;
end;
```

É claro que existem inúmeras formas de fazer isso e já foram discutidas várias técnicas a respeito do assunto em vídeo aulas e artigos elaborados por outros colegas. Minha recomendação é que faça tal configuração utilizando arquivos externos com os parâmetros, como um arquivo INI, por exemplo.

Por didática, deixo a critério do leitor a responsabilidade de implementar da melhor forma possível a conexão com o banco de dados. Para este exemplo apenas não esqueça de mudar, se necessário, as propriedades *LibraryName*, *VendorLib* e o parâmetro *Database* onde encontra-se o diretório da base de dados.

Aplicando o Patch

Digamos que o coração da nossa classe começa a bater aqui. Nesse ponto, vamos desenvolver a principal função: *Execute*. Pesquisando na internet encontrei diversas soluções para aplicar alterações, atualizações, inclusões e exclusões no banco de dados.

Coletei todas, fiz algumas adaptações e eis a função na **Listagem 6** e a seguir alguns comentários.

O primeiro passo do *Execute* é testar se a conexão com o banco foi bem sucedida e logo em seguida, testamos novamente se o arquivo de *Patch* existe no local indicado. O próximo trecho faz a leitura do arquivo de *patch* usando *AssignFile*.

Listagem 6. Função Execute da classe TPatche

```
function TPatches.Execute: Boolean;
var
  ArqAux: TextFile;
  Log: TextFile;
  LogTxt: string;
  i: Integer;
  Lin: string;
  FraseSQL: string;
  Conjunto: TStringList;
  Transacao: TTransactionDesc;
begin
  Result := False;
  if not ConectarAoBanco then
    Exit;
  if not FileExists(FPatch) then
  begin
    MessageBox(0, PChar('O arquivo de patch não '+
      'existe ou o seu caminho está incorreto.' +
      #13#13 + FPatch), 'Erro', MB_ICONERROR);
    Exit;
  end;
  Conjunto := TStringList.Create;
  FraseSQL := '';
  AssignFile(ArqAux, FPatch);
  reset(ArqAux);
  while not EOF(ArqAux) do
  begin
    ReadLn(ArqAux, Lin);
    FraseSQL := FraseSQL + Lin;
    if Pos(';', FraseSQL) > 0 then
    begin
      Conjunto.Add(FraseSQL);
      FraseSQL := '';
    end
    else
      FraseSQL := FraseSQL + ' ';
  end;
  CloseFile(ArqAux);
  { Cria ou abre o arquivo de Log }
  LogTxt := ExtractFilePath(Application.ExeName) +
    Copy(ExtractFileName(FPatch), 1,
    Length(ExtractFileName(FPatch))-6) + '.log';
  AssignFile(Log, LogTxt);
  FileMode := $0002;
  ReWrite(Log);
  { Para cada linha do StringList, executa o comando }
  for i := 0 to Conjunto.Count - 1 do
  begin
    try
      Transacao.TransactionID := 1;
      Transacao.IsolationLevel := xilREADCOMMITTED;
      FConexao.StartTransaction(Transacao);
      FConexao.ExecuteDirect(Conjunto[i]);
      FConexao.Commit(Transacao);
      WriteLn(Log, Copy(ExtractFileName(FPatch), 1,
        Length(ExtractFileName(FPatch)) - 6) + ': ' +
        ' Sucesso: ' + Trim(Conjunto[i]));
      Result := True;
    except on E: Exception do
      begin
        { Caso ocorra um erro, da um Roolback e
          atualiza o arquivo de log }
        FConexao.Rollback(Transacao);
        WriteLn(Log, Copy(ExtractFileName(FPatch), 1,
          Length(ExtractFileName(FPatch)) - 6) + ': ' +
          ' Erro:    ' + Trim(Conjunto[i]));
        WriteLn(Log, '> > >  ' + E.Message);
        Continue;
      end;
    end;
  end;
  Desconectar;
  CloseFile(Log);
end;
```

Então percorremos o arquivo a fim de armazenar linha a linha os comandos que serão executados, ou seja, em cada linha de um *StringList* (variável *Conjunto*) haverá um comando. Note que a variável *FraseSQL* recebe o conteúdo da linha atual durante o *loop* e quando a ocorrência do ponto-e-vírgula é encontrada a mesma é adicionada ao *StringList*.

Isso faz com que tenhamos apenas um comando por linha em *Conjunto*. Imagine instruções como *Create Table*, por exemplo, onde, normalmente, escrevemos a instrução SQL em mais de uma linha. Essa será transformada em apenas uma pra ser executada.

Tive também o cuidado de criar um arquivo de *log*, para identificar possíveis comandos não executados e para manter um relatório final da atualização. O *log* é criado por *patch*, ou seja, termos um arquivo LOG para cada *patch* informando que comandos tiveram êxito e os que falharam.

Basicamente a parte mais importante vem a seguir. É onde efetivamente aplicamos linha a linha os comandos coletados. A explicação é simples:

- Implementamos um *loop* em nosso conjunto de comandos (*Conjunto*);
- Envolvemos um comando em um bloco *try..except..end*;
- Na primeira fase, criamos uma transação, indicamos o nível de isolamento da mesma e a iniciamos;
- Logo em seguida o comando contido na leitura de *Conjunto[i]*, ou seja, o comando atual do *loop* é aplicado usando o *ExecuteDirect* do *SQLConnection* e executamos um *Commit*.

Caso tudo ocorra bem escrevemos no *log* o nome do *patch*, sucesso ou fracasso e o comando. Ocorrendo algum erro aplicamos um *Rollback* na transação, abortando o último comando aplicado e novamente escrevemos no *log* a ocorrência, mas dessa vez indicando também a descrição do erro.

Chamamos *Continue* para que o *loop* continue normalmente até que *Execute* termine. Dessa forma, não perdemos as alterações anteriores e nem paramos o processo no meio. Por fim, desconectamos do banco e liberamos o arquivo de *log*.

Pronto, nossa classe está terminada e agora basta implementar sua chamada no formulário principal.

Usando a classe

Basicamente vamos codificar pouco agora. Clique duas vezes no *Aplicar* e em seu evento *OnClick* digite o código da **Listagem 7** onde encontra-se toda a chamada.

Listagem 7. Evento OnClick do Aplicar

```
uses cPatche;
...
procedure TfrmAtualizador.btnAplicarClick(
  Sender: TObject);
var
  Patchs: TPatches;
  i: Integer;
begin
  flbLogs.Directory := ExtractFilePath(
    Application.ExeName);
  flbLogs.Mask := '*.log';
  flbLogs.Update;
  pgbArquivos.Position := 0;
  pgbArquivos.Max := flbPatches.Count;
  if flbPatches.Count = 0 then
    MessageDlg('Não existem patches para serem '+
      'aplicados.', mtInformation, [mbOk], 0);
  Patchs := TPatches.Create;
  for i := 0 to flbPatches.Count - 1 do
  begin
    Patchs.Patch := ExtractFilePath(
      Application.ExeName) + flbPatches.Items[i];
    lblAtualizacoes.Caption := 'Aplicando patch ' +
      flbPatches.Items[i];
    Patchs.Execute;
    pgbArquivos.Position := pgbArquivos.Position + 1;
  end;
  lblAtualizacoes.Caption := '';
  MessageDlg('Script de atualização executado.',
    mtInformation, [mbOk], 0);
  btnAplicar.Enabled := False;
  flbLogs.Directory := ExtractFilePath(Application.ExeName);
  flbLogs.Mask := '*.log';
  flbLogs.Update;
end;
```

Listagem 8. Função para limpeza e inicialização de variáveis.

```
procedure TfrmAtualizador.FormActivate(
  Sender: TObject);
var
  i: Integer;
begin
  flbLogs.Directory := ExtractFilePath(
    Application.ExeName);
  flbLogs.Mask := '*.log';
  for i := 0 to flbLogs.Items.Count - 1 do
    DeleteFile(ExtractFilePath(Application.ExeName) +
      flbLogs.Items[i]);
  pgbArquivos.Max := 0;
  pgbArquivos.Position := 0;
  lblAtualizacoes.Caption := '';
  flbPatches.Directory := ExtractFilePath(
    Application.ExeName);
  flbPatches.Mask := '*.patch';
  btnAplicar.Enabled := True;
end;
```

Figura 2. Processo finalizado

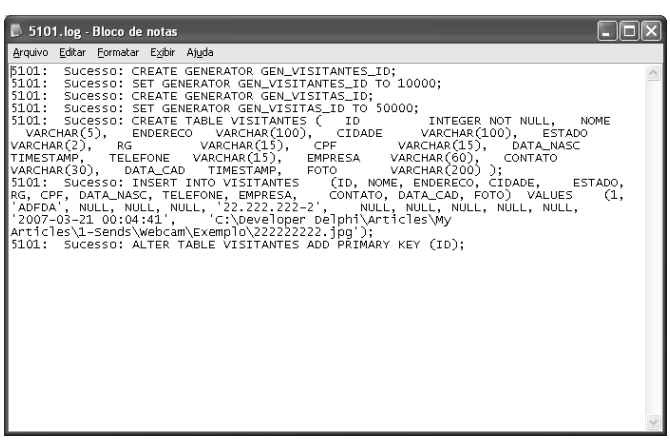


Figura 3. Exemplo de arquivo de log

O primeiro bloco consiste em atualizar o *FileLisBox* de logs (*flbLogs*) com o diretório da aplicação onde serão gerados os *logs* e também atualizar o valor do *Position* e *Max* do *ProgressBar*. Em seguida verificamos se existem arquivos de *patch* para serem aplicados e iniciamos um *loop* nos itens do *flbPatches*. Durante o *loop* atualizamos a propriedade *Patch* da classe com o caminho do *script* atual e então chamamos o *Execute*.

Na **Listagem 8** encontra-se o código do evento *OnActivate* do formulário principal, onde zeramos e inicializamos algumas variáveis.

Na **Figura 2** podemos ver a mensagem final de todo o processo. E na **Figura 3** um exemplo de *log* gerado pela classe.

Algumas considerações

As execuções de *scripts* relativamente grandes podem causar "congelamentos" no programa por um longo período, já que cada linha de comando deve ser executada separadamente. Recomendo que na implementação dessa classe sejam também inclusas funções, procedimentos e/ou propriedades capazes de atualizar objetos passados como parâmetro para indicar ao usuário o que o sistema está fazendo no momento, evitando assim que esse usuário acredite que o programa travou.

Outra prática que deve ser aplicada é a verificação antecipada do *patch* na base de dados atual, pra evitar que o mesmo arquivo de *script* seja aplicado mais de uma vez resultando *crash* no sistema.

Conclusão

Como falado no início deste artigo, a saúde do banco de dados é uma das principais preocupações que o desenvolvedor deve ter em relação ao sistema. Manter o banco sempre atualizado e compatível com a versão do programa reduz o número de suporte e ajuda no desenvolvimento de novas soluções para o dia a dia.

Neste artigo vimos como criar uma ferramenta relativamente simples para auxiliar na atualização do banco de dados. Cabe ao desenvolvedor cuidar do maior bem da aplicação, o banco. ■

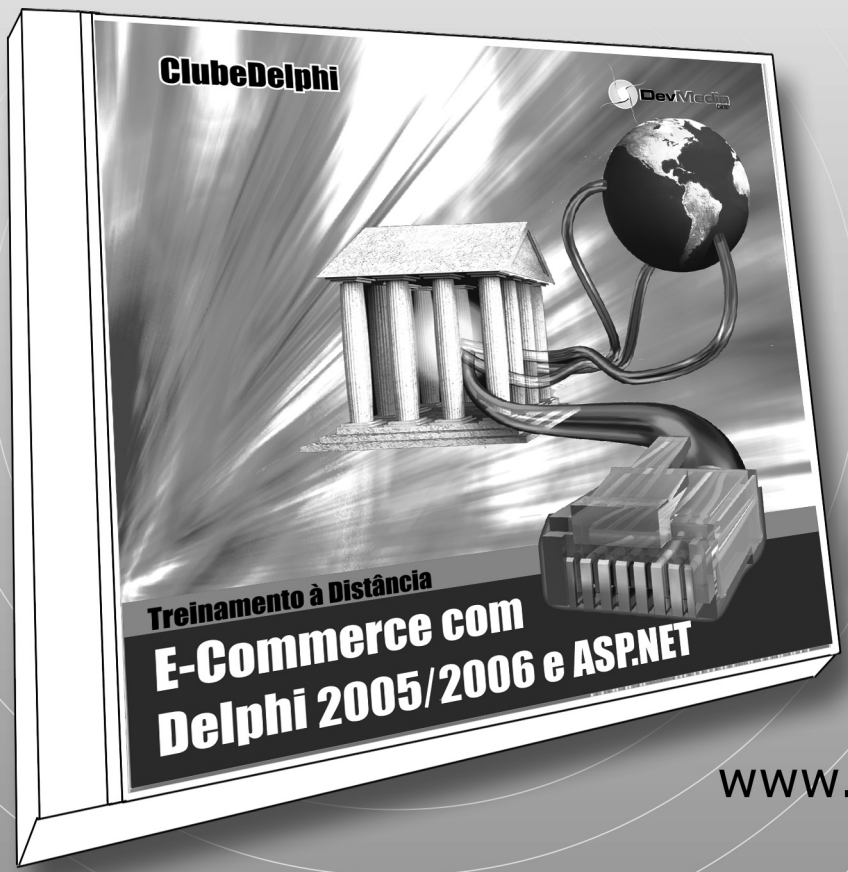
Links

Preparando o sistema para mais de um banco (Adriano Santos)

www.devmedia.com.br/articles/viewcomp.asp?comp=2245

Treinamentos a distância.

Aprenda de forma prática, eficiente e econômica.



E-Commerce com Delphi 2005/2006 e ASP.NET

Aprenda a desenvolver poderosas aplicações Web através do Delphi 2005/2006 e as tecnologias ASP.NET, WebBroker, Intraweb e WebSnap.
O curso conta com módulos textuais;
35 horas de vídeo-aulas;
fórum exclusivo para os alunos e suporte on-line.

COMPRE JÁ O SEU!
www.devmedia.com.br/curso

Outros Cursos

Delphi e Relatórios

Com este curso você desenvolverá relatórios através do Delphi 7, do Delphi 2005 e Delphi 2006, utilizando as ferramentas Rave Reports, Quick Reports e Crystal Report. Além do módulo textual o curso contém mais de 20 horas de vídeo-aulas; fórum exclusivo e suporte

Evento à distância - Introdução ao .NET

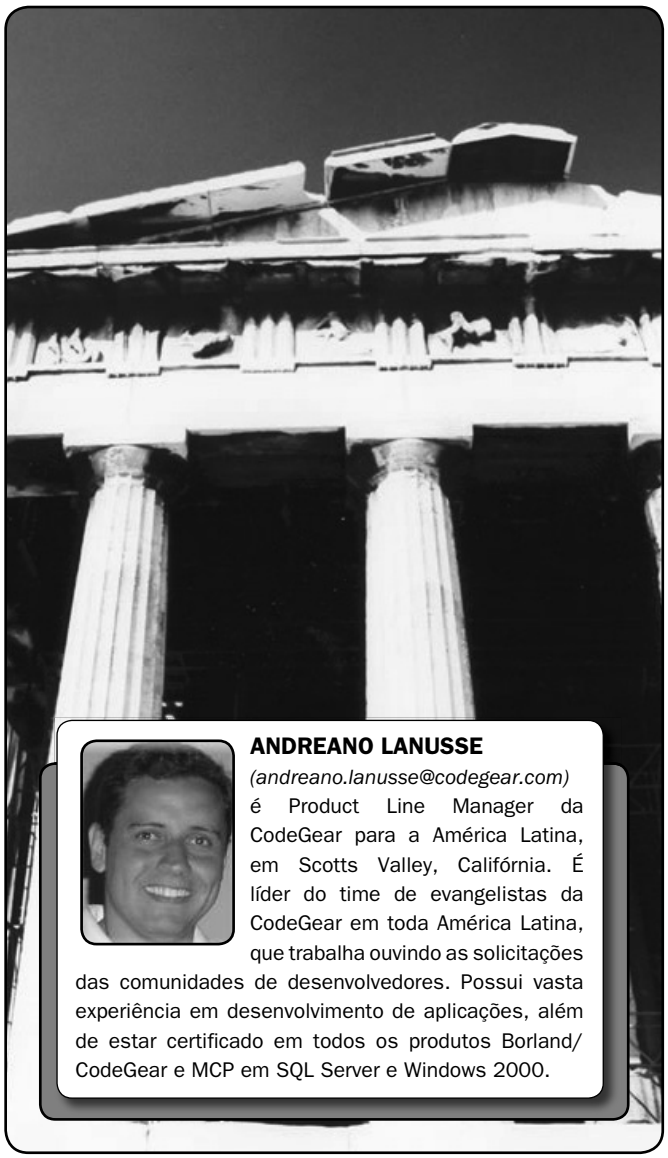
Conheça os recursos do ASP.NET com o Visual Studio 2003 e aprenda a criar páginas ASP.NET; Webservices; trabalhar com a IDE do VS; componentização; DataGrid e criar aplicações de BD com o ADO.NET! São 6h de palestras em formato de vídeo-aulas e você pode tirar suas dúvidas com os palestrantes!

Evento à distância - Novidades do Delphi 2005

Conheça os recursos do Delphi 2005 e aprenda a criar páginas ASP.NET; Webservices; trabalhar com a nova IDE do Delphi; utilizar programação OO no Delphi 2005; trabalhar em equipe com o novo Starteam e criar aplicações de BD com o ADO.NET!

DevMedia

Delphi 2007 for Win32



ANDREANO LANUSSE
(andreano.lanusse@codegear.com)
é Product Line Manager da CodeGear para a América Latina, em Scotts Valley, Califórnia. É líder do time de evangelistas da CodeGear em toda América Latina, que trabalha ouvindo as solicitações das comunidades de desenvolvedores. Possui vasta experiência em desenvolvimento de aplicações, além de estar certificado em todos os produtos Borland/ CodeGear e MCP em SQL Server e Windows 2000.

Caros Delpheiros, desde o anúncio da CodeGear, a nova unidade de negócios da Borland focada 100% nos desenvolvedores, lançamos três novos produtos: o JBuilder 2007 para desenvolvimento Java, o Delphi for PHP, a inovação para os desenvolvedores PHP que passam a contar com a VCL para PHP, e também o Delphi 2007 for Win32, que é a resposta ao desenvolvimento nativo de aplicações para Win32.

E é sobre esse lançamento, o Delphi 2007 for Win32, que trataremos neste artigo, assim como todas as novidades dessa versão. Como o próprio nome diz, trata-se de um IDE focado em desenvolvimento para a plataforma Win32, que deverá ser um marco na história de desenvolvimento com Delphi da mesma forma que foi o Delphi 7.

Por que uma versão de Delphi para Win32?

A resposta para essa pergunta é muito simples, porque a maioria dos desenvolvedores Delphi pediu um IDE mais leve para continuar desenvolvendo para Win32. Muitos não estão utilizando .NET e os que desenvolvem no IDE, usam apenas para ASP.NET.

O WinForms não tem tido grande aceitação entre os desenvolvedores Delphi. Isso está muito claro, já que o Delphi traz a VCL que está aprovada e ainda oferece muito mais recursos e componentes ao desenvolvimento desktop.

Mas isso não quer dizer que não teremos Delphi para .NET, não só teremos como já estamos trabalhando na nova versão do BDS que se chamará RAD Studio. A próxima versão do C++ também está em desenvolvimento e chega ainda esse ano, junto com grandes inovações.

Acompanhe essas e outras novidades pelo *Developer Network Brasil* (*dn.codegear.com/br*), pelo blog da CodeGear (*blogs.codegear.com*) e no site da revista ClubeDelphi.

Instalação do Delphi 2007 para Win32

O Delphi 2007 tem alguns pré-requisitos para sua instalação, com os quais você não precisa se preocupar, já que o IDE verificará se eles estão instalados em sua máquina e os instalará caso você não os tenha.

O Delphi 2007 utiliza .NET 2.0, mas as aplicações geradas não necessitam de .NET. Na **Tabela 1** está a lista de pré-requisitos que o Delphi 2007 verificará e instalará se necessário.

Qualidade e Performance

Continuamos na busca de mais qualidade e muito mais performance com o Delphi 2007. Com o BDS 2006, por exemplo, conseguimos mais performance do que com o Delphi 7. Agora, imagine o Delphi 2007 com apenas Win32.

Pois enquanto eu escrevia este artigo, fiz um teste na minha máquina, que é um Dell Pentium M 1.8GHZ com 2GB de RAM, e meu Delphi 2007 carregou em apenas 10 segundos!

TeeChart 7

Um dos componentes mais utilizados pelos desenvolvedores Delphi, o *TeeChart*, agora está na versão 7 no Delphi 2007, trazendo novos recursos para trabalhar com gráficos (**Figura 1**).

Rave Report 7.5

O gerador de relatórios Rave Report traz a versão 7.5, com diversas atualizações e é mais uma opção de desenvolvimento de relatórios para os usuários que estavam em Delphi 6 ou numa versão anterior.

Novidades do IDE

O IDE possui uma série de novidades, entre elas:

• *MSBuild*: você poderá controlar o que acontece antes e depois da compilação do projeto, isso é excelente para quem quer automatizar a compilação. Ao abrir um projeto, o Delphi fará a conversão automaticamente para a extensão DPROJ para suportar o *MSBuild*;

• *Vista e XP Themes*: o IDE suporta os temas para Windows Vista e XP. Agora é *default* e você pode habilitar e desabilitar através das opções de projeto;

• *Project Manager*: suporta múltiplas seleções e com isso permite abertura, gravação e remoção de vários arquivos simultaneamente;

• *File Browser*: você não precisa mais abrir o Windows Explorer ou ir em *File Open* toda vez para abrir um arquivo, o *File Browser* fornece acesso direto à sua árvore de diretórios de dentro do IDE.

dbExpress Framework

Uma das grandes novidades no Delphi 2007 é a completa reestruturação da arquitetura do dbExpress, que agora conta com os drivers escritos 100% em Delphi. Fizemos alguns testes de performance em nossos laboratórios, simulando situações diversas, envolvendo vários bancos de dados e obtivemos ganhos de até 100% de performance.

dbExpress 4 também é um marco para aplicações que requerem conectividade com banco de dados, pois a nova arquitetura está preparada para Win32 e para .NET. Atualmente, os componentes dbExpress são o novo padrão para desenvolvimento .NET com Delphi e estará disponível para aplicações Windows Forms e ASP.NET no RAD Studio (**Figura 2**).

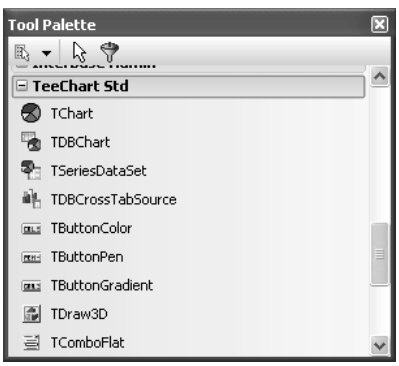


Figura 1. Componentes do TeeChart 7

ClubeDelphi PLUS

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Adriano Santos que mostra algumas das novidades do Delphi 2007.

www.devmedia.com.br/articles/viewcomp.asp?comp=5079

ClubeDelphi PLUS

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Adriano Santos que mostra como instalar a versão trial do Delphi 2007.

www.devmedia.com.br/articles/viewcomp.asp?comp=5002

Pré-requisito	Descrição
Microsoft .NET Framework Version 2.0 Redistributable Package	O IDE apóia-se no .NET 2.0 Framework para alguns recursos como *MSBuild* e outros serviços essenciais.
.NET Framework 2.0 Software Development Kit (SDK)	O produto apóia-se no SDK para serviços essenciais como *HelpViewer*.
Microsoft Visual J# Version 2.0 Redistributable Package	Obrigatório para a maioria das funcionalidades do Together no IDE.
Microsoft Data Access Components (MDAC) 2.8	Obrigatórios para os componentes dbGo.
Microsoft Core XML Services (MSXML) 6.0	Obrigatório para a página *Welcome* e outros serviços de processamentos XML do IDE.

Tabela 1. Pré-requisitos de instalação do Delphi 2007

É importante lembrar que as aplicações desenvolvidas nas versões anteriores do Delphi são 100% compatíveis com Delphi 2007. Mudamos apenas a infra-estrutura do dbExpress. O DBX Framework é o novo conjunto de classes que facilita o acesso a bases de dados.

Você poderá ter total acesso às informações armazenadas no banco de dados através dessas classes e não apenas através dos componentes *SQLConnection*, *SQLDataSet*, *SQLQuery* e outros.

dbExpress 4 - Drivers

Novos drivers foram adicionados ao dbExpress: InterBase 2007, MySQL 4.1 e MySQL 5.0. Os drivers para Oracle, InterBase e MySQL passam a suportar caractere *unicode*. Novos conceitos que chamamos de *Delegate Driver* e *Pools Connections* estão disponíveis no dbExpress.

DelegateDriver permite que você conecte uma classe ao seu *tracer* e, através dela, possa capturar as informações que trafegam por sua conexão, funcionando como uma espécie de SQL Monitor, com a diferença de que agora você pode criar sua classe, extender a classe *DBXTrace* e manipular as informações de sua maneira.

Ao imaginar uma solução de replicação, perceberá que o *Delegates* pode ajudar muito. Acompanhe no código da **Listagem 1** o resultado de um *log* gerado pelo *Delegate* em Delphi Language. É mais fácil de ler, entender e até mesmo de executar novamente.

A configuração de um *trace*, como no exemplo do código a seguir, capturará os eventos de acordo com parâmetro *Trace Flags* e irá gravá-los no arquivo *c:\dbxtrace.txt*. Sua conexão dbExpress terá um parâmetro indicando as configurações do *trace*, por exemplo: *DelegateConnection= DBXTraceConnection*.

```
[DBXTraceConnection]
DriverName=DBXTrace
TraceFlags=EXECUTE;COMMAND;CONNECT
TraceDriver=true
TraceFile=c:\dbxtrace.txt
```

Você também pode utilizar, nativamente, *pools* de conexão com dbExpress. Na **Listagem 2** temos um *alias* (*Pool_DelegateDemo*) do dbExpress, delegando ao *DBXPoolConnection* o controle de *pool* de conexões, onde está configurado um número máximo de 16 conexões.

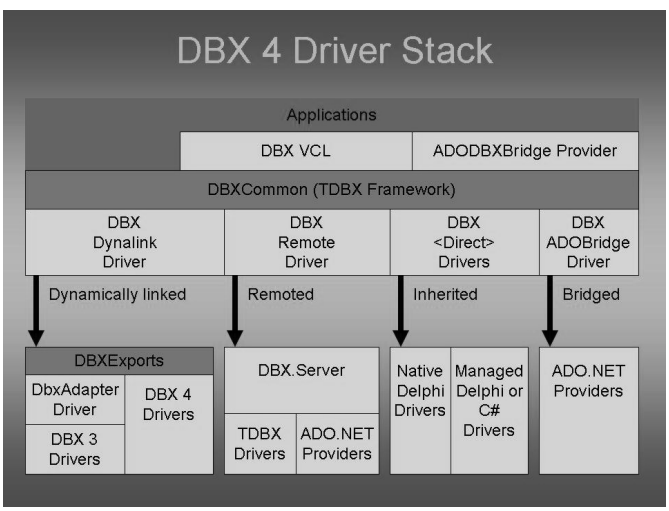


Figura 2. Arquitetura do dbExpress para suportar Win32 e .NET

dbExpress 4 - VCL

Algumas alterações foram necessárias nos componentes do dbExpress, visando facilitar o desenvolvimento, mantendo sua aplicação compatível. Caso queira conhecer todas as alterações nos componentes, o help do Delphi 2007 traz todas essas informações (*ms-help://borland.bds5/devnet/dbexpressframeworkcompat_xml.html*).

Na **Listagem 3** temos um exemplo de uma aplicação *Console* que utiliza recursos de conexão com banco de dados, lê parâmetros da conexão, executa *query*, lê o resultado de uma *query* e utiliza transação.

Debugger

A inclusão e alteração de diversos recursos foi incorporada ao Delphi 2007 for Win32 e todos visando facilitar a vida do desenvolvedor. Agora é possível configurar a barra de rolagem de tela no *log* de eventos e abrir separadamente cada item da janela de CPU ao arrastar a janela de *debugger* e pressionar CTRL. Dessa forma, ela se torna transparente, facilitando a leitura do código durante o *debug*.

VCL e Windows Vista

Todas as aplicações compiladas em Delphi 2007 funcionarão em Windows Vista, portanto, ao recompilar a sua aplicação,

Listagem 1. Log gerado por um Delegate

```
Log Opened ========================================
{CONNECT} ConnectionC1.Open;
{COMMAND} CommandC1_1 := ConnectionC1.CreateCommand;
{COMMAND} CommandC1_1.CommandType := 'Dbx.SQL';
{COMMAND} CommandC1_1.CommandType := 'Dbx.SQL';
{COMMAND} CommandC1_1.Text := 'select * from employee';
{PREPARE} CommandC1_1.Prepare;
{COMMAND} ReaderC1_1_1 := CommandC1_1.ExecuteQuery;
{COMMAND} CommandC1_2 := ConnectionC1.CreateCommand;
{COMMAND} CommandC1_2.CommandType := 'Dbx.MetaData';
{COMMAND} CommandC1_2.Text := 'GetIndexes   "localhost:C:\CodeGear\
                              InterBase\examples\database\"+
                              "employee.ib"."sysdba"."employee" ';
{COMMAND} ReaderC1_2_1 := CommandC1_2.ExecuteQuery;
{READER}  { ReaderC1_2_1 closed.  6 row(s) read }
{READER} FreeAndNil(ReaderC1_2_1);
```

Listagem 2. Pools de conexão com o dbExpress

```
[DBXPoolConnection]
DriverName=DBXPool
MaxConnections=16
MinConnections=0
ConnectTimeout=0

[Pool_DelegateDemo]
DelegateConnection=DBXPoolConnection
DriverName=Interbase
DriverUnit=DBXDynalink
DriverPackageLoader=TDBXDynalinkDriverLoader
DriverPackage=DBXCommonDriver110.bpl
DriverAssemblyLoader=Borland.Data.TDBXDynalinkDriverLoader
DriverAssembly=Borland.Data.DbxCommonDriver,
  Version=11.0.5000.0,Culture=neutral,
  PublicKeyToken=a91a7c5705831a4f
Database= localhost:C:\CodeGear\InterBase\examples\database\employee.ib
RoleName=RoleName
User_Name=sysdba
Password=masterkey
ServerCharSet=
SQLDialect=3
BlobSize=-1
CommitRetain=False
WaitOnLocks=True
ErrorResourceFile=
LocaleCode=0000
Interbase TransIsolation=ReadCommited
Trim Char=False
```

ela já estará compatível. A VCL do Delphi 2007 foi atualizada para suportar Windows Vista, além da incorporação de novos componentes específicos.

Os novos componentes são: *FileOpenDialog, FileSaveDialog* e *TaskDialog*. E também diversas classes foram criadas, como:

TCustomFileDialog, TCustomFileOpenDialog, TCustomFileSaveDialog, TCustomTaskDialog, TFavoriteLinkItem, TFavoriteLinkItems, TFavoriteLinkItemsEnumerator, TFileTypeItem, TFileTypeItems, TTaskDialogBaseButtonItem, TTaskDialogButtonItem, TTaskDialogButtons, TTaskDialogButtonsEnumerator, TTaskDialogProgressBar e *TTaskDialogRadioButtonItem*

Além disso, os componentes referentes à abertura e gravação de arquivos passam a ter a mesma interface do Windows Vista. E se sua aplicação for executada em Windows XP? Não se preocupe, pois a VCL sabe que forma apresentar.

A função *TaskMessageDlg* foi criada para suportar o Windows Vista. Ao executar em XP, a função *MessageDlg* será executada automaticamente. Quando recebe *True*, a variável global *UseLatestCommonDialogs* causa a chamada para a caixa de diálogo equivalente no Windows Vista nos componentes *OpenDialog*, *SaveDialog*, *OpenPictureDialog* e *SavePictureDialog*. Por exemplo, o *Open Dialog* em Windows Vista é semelhante a **Figura 3**. Já o *SaveDialog* ficará semelhante a **Figura 4**.

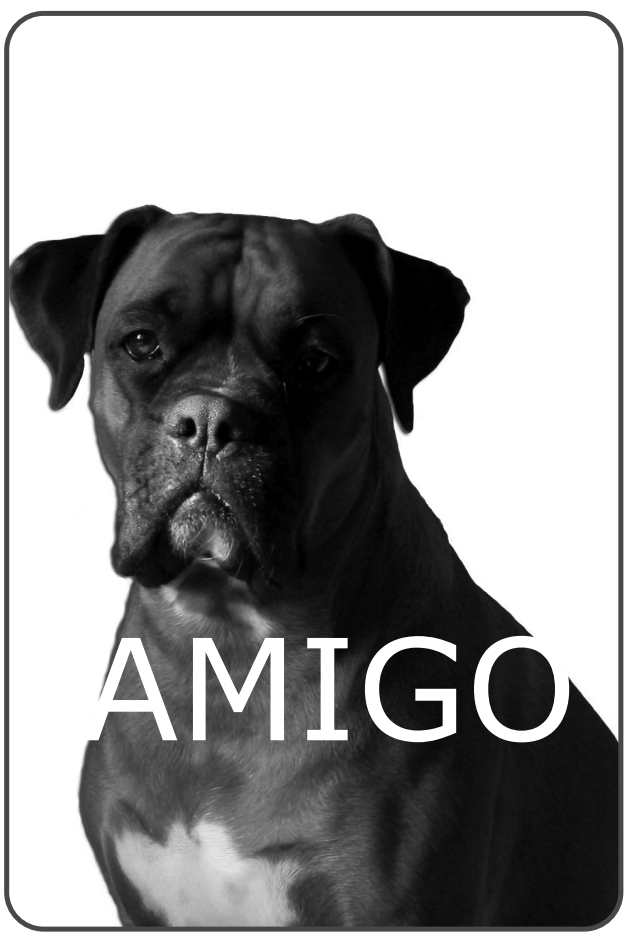
As units a seguir sofreram alterações para suportar as novas API's do Windows: *UxThemes* (nova), *DwnApi* (nova), *ActiveX* (atualizada), *Windows* (atualizada), *Messages* (atualizada), *CommCtrl* (atualizada) e *ShlObj* (atualizada).

VCL for Web

Outra novidade no Delphi 2007 é a VCL para Web, que está baseada no IntraWeb 9 e traz total integração com Ajax. O fato é que integrar Ajax no desenvolvimento sem um framework é muito improdutivo, já que você passa grande parte do tempo escrevendo *JavaScript*.

Com a VCL for Web, tudo isso ficou mais fácil, pois os principais componentes têm evento *Async*, o que permite escrever seu código Delphi e executá-lo sem fazer o *Submit* da página. Por exemplo, o *IWButton* tem vários eventos e o *Click* no botão pode ser feito de duas formas:

- *Utilizando Ajax com OnAsyncClick*: o que você executar através desse evento será utilizando Ajax;
- *Sem utilizar Ajax com OnClick*: o tradicional evento permanece, caso você queira continuar utilizando chamadas convencionais.



Existem coisas
que não conseguimos
ficar sem!

...só pra lembrar,
sua assinatura pode
estar acabando!

Renove Já!

www.devmedia.com.br/renovacao

Para mais informações:
www.devmedia.com.br/central

DevMedia GROUP

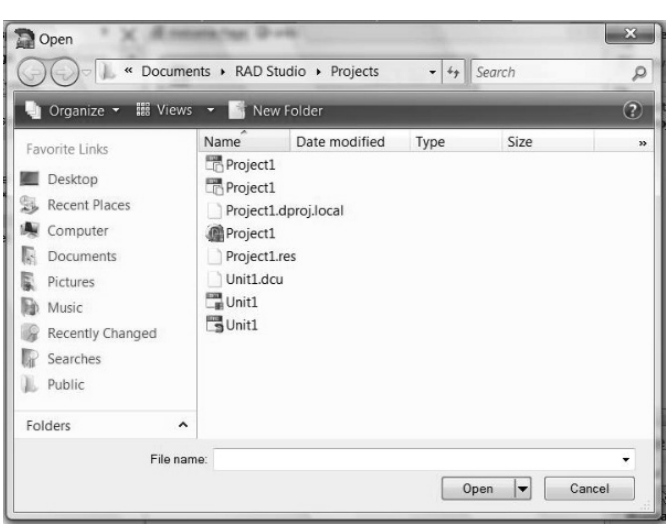


Figure 3. OpenDialog em Windows Vista

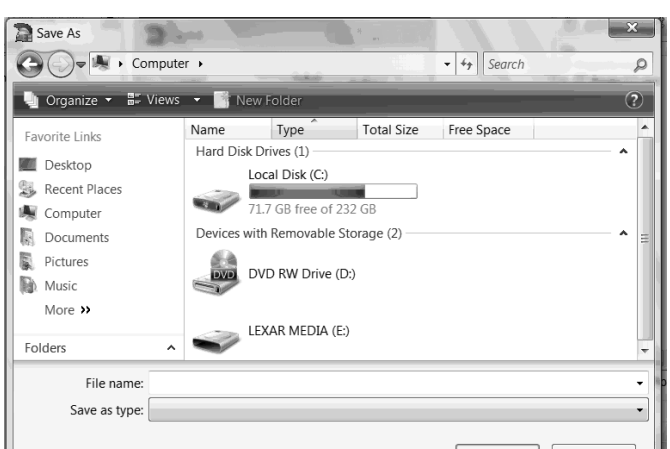


Figure 4. SaveDialog em Windows Vista

Listagem 3. Projeto Console com conexão a banco de dados

```
program DBX4Example;

{$APPTYPE CONSOLE}
uses
  SysUtils, DBXDynalink, Dialogs, DBXCommon;
var
  aConnName: string;
  aDBXConn: TDBXConnection;
  aDBXTrans: TDBXTransaction;
  aCmnd: TDBXCommand;
  aReader: TDBXReader;
  i, aColCount: integer;
begin
  aDBXConn := TDBXConnectionFactory.GetConnectionFactory.
    GetConnection('EMPLOYEE','sysdba','masterkey');
  { Write the all connection parameters }
  Writeln('======== Connection Properties ========');
  WriteLn(aDBXConn.ConnectionProperties.Text);
  Writeln('=======================================');
  Writeln('');
  if aDBXConn <> nil then
  begin
    aCmnd := aDBXConn.CreateCommand;
    { Start transaction }
    aDBXTrans:= aDBXConn.BeginTransaction(
      TDBXIsolations.ReadCommitted);
    for I := 1 to 100000 do
    begin
      { Prepare and execute the SQL Statement }
      aCmnd.Text := 'SELECT * FROM Country';
      aCmnd.Prepare;
      aReader := aCmnd.ExecuteQuery;
      aColCount := aReader.ColumnCount;
      Writeln('Results from Query: ' + aCmnd.Text);
      Writeln('Number of Columns:  ' + IntToStr(aColCount));
      while aReader.Next do
      begin
        Writeln(aReader.Value['Country'].GetAnsiString);
      end;
      Writeln( '================================');
      Writeln( '' );
    end;
      { Commit transaction }
      aDBXConn.CommitFreeAndNil(aDBXTrans);
      Readln;
      aReader.Free;
      aCmnd.Free;
      aDbxConn.Free;
  end;
end.
```

Conclusão

Este artigo trouxe uma visão geral das novidades do Delphi 2007 for Win32 e o compromisso da CodeGear em continuar suportando a plataforma Win32, apresentando suporte ao Windows Vista. Os desenvolvedores Delphi agora podem compatibilizar suas aplicações com o novo sistema operacional da Microsoft através de uma simples mudança, já que a VCL está 100% compatível com a novidade.

Nesse mês, a CodeGear também lançou o Delphi for PHP. Com essa novidade, a companhia reforça seu objetivo de estar 100% focada nos desenvolvedores, trazendo sempre soluções inovadoras.

Tanto o Delphi 2007 for Win 32 quanto o Delphi for PHP já estão disponíveis para venda no Brasil e você pode adquiri-los através do Borland Shop (*www.borlandshop.com.br*). Mais novidades vem por aí. Até o próximo artigo. ■

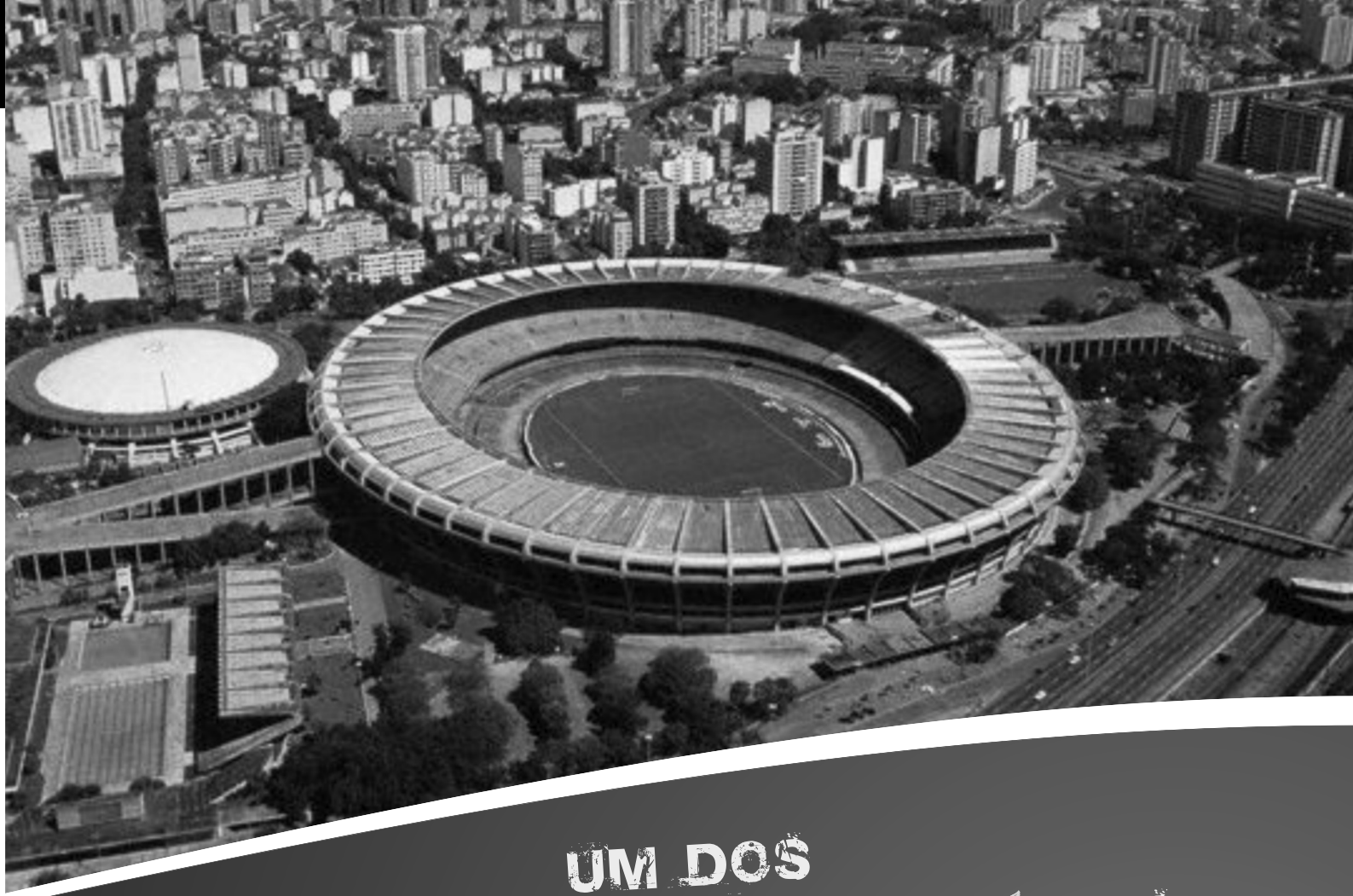
Links

Blog do autor
blogs.codegear.com/andreanolanusse

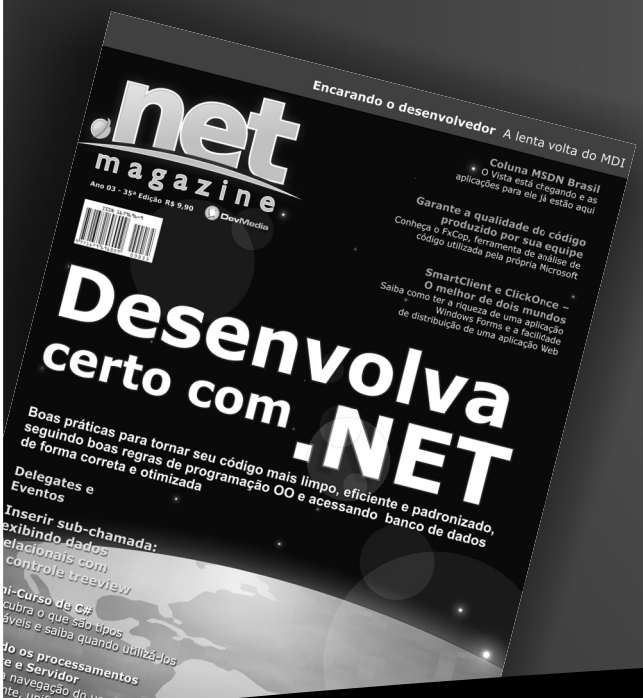
Treinamentos via Web sobre os produtos CodeGear como o Delphi estão disponíveis
www.borlandopenuniversity.com.br

Site de relacionamento da CodeGear com os desenvolvedores
dn.codegear.com/br

Site oficial da CodeGear
www.codegear.com



UM DOS MAIORES ESTÁDIOS DO MUNDO. FEITO POR BRASILEIROS.



Após três anos de publicação decidimos encerrar na edição 35 o uso da marca MSDN Magazine no Brasil. A **MSDN Magazine** agora se chama **.NET Magazine**, uma publicação que tem o orgulho de ser **100% brasileira!** Com a mudança a revista ganhou mais qualidade, mais conteúdo, **mais nacionalidade.**

.NET Magazine
A revista do desenvolvedor que tem orgulho de ser brasileiro.

.net magazine

Já nas bancas!

Desenvolvendo uma Loja Virtual com ASP.NET, UML e ECO – Parte 2

PAULO ROBERTO QUICOLI
(pauloquicoli@gmail.com)
é analista e programador da Control-M Informática. Trabalha com Delphi, desde sua primeira versão, e Firebird desenvolvendo aplicações cliente/servidor. Formado em Tecnologia de Processamento de dados pela FATEC, na cidade de Taquaritinga/SP

Na edição anterior, foi apresentada uma introdução ao ECO e iniciamos um site de compras, o *ECOStore*. Neste artigo continuando com nosso objetivo, vamos implementar o cadastro de Produtos e oferecer uma forma de localizá-los.

Cadastro de Categorias

Nossos produtos possuem Categorias, portanto é preciso cadastrá-las inicialmente. Abra o projeto que criamos no artigo anterior e crie um novo *ECO WebForm*, acessando o menu *File>New>Other>New ASP.NET Files>Eco ASP.NET Page*. Renomeie-o como "CadCategoria.aspx".

Ligue o *RootHandle* ao *ECOSpace* através da propriedade *EcoSpaceType*. Adicione no formulário o *User Control* da aplicação. Vamos adicionar agora um *ExpressionHandle*, da aba *Enterprise Core Objects* da *Tool Palette*, e o renomeie para "exphCategoria".

Através de um *ExpressionHandle* podemos obter o *ECOSpace* de determinados objetos e de determinadas classes. Alterando a propriedade *RootHandle* para o objeto *RootHandle* constante da página, ligamos o *exphCategoria* ao *ECOSpace*.

Compile o projeto. Para definir que objetos e classes serão retornados, utilizamos uma expressão OCL que será especificada na propriedade *Expression*. Clicando no botão existente na propriedade, o editor OCL é exibido, conforme a **Figura 1**.

Metodologia

A aplicação desenvolvida neste mini-curso é desenvolvida sobre um modelo multicamadas, onde temos: banco de dados, aplicação web, camada de persistência e interface (browser). As vantagens dessa abordagem são muitas: separação lógica entre as partes da aplicação, facilidade de manutenção, reutilização de código, melhor uso de técnicas de POO, torna a lógica de negócio independente da arquitetura utilizada e tecnologias empregadas (ex.: ASP.NET, Windows Forms, Web Services), menor impacto na mudança de requisito e redução de custos.

Veja que editor já exibe as classes presentes no modelo. Dê um duplo clique sobre *Categoria*. Será habilitada uma lista de expressões que podem ser aplicadas à classe. Dessa lista, escolha *allInstances* dando um duplo clique sobre ela e pressione o botão OK (**Figura 2**).

Adicione ao formulário um *DataGrid* e escolha um visual para ele (através da opção *AutoFormat*, presente no *Object Inspector*). Através desse controle, vamos adicionar, editar e excluir as categorias dos produtos.

Diferentemente de uma aplicação Windows Forms, um *DataGrid* para ASP.NET exige a existência de um campo chave para que seja possível editar uma linha da lista de objetos.

Em uma aplicação ECO, essa chave é representada através da propriedade *ExternalID* que identifica de forma única um objeto dentro do *ECOSpace*, seria algo similar a uma chave primária, que no caso do ECO é aplicada à toda coleção de objetos e não apenas uma determinada classe.

Por padrão a propriedade *ExternalID* é desabilitada para uso externo, permanecendo oculta durante o tempo de projeto e sendo utilizada apenas internamente pelo framework. Nesse exemplo, como precisamos dela, podemos habilitar sua utilização.

Nas propriedades do *exphCategoria* veja que existe uma chamada *AddExternalID*. Altere seu valor para *True*. Acesse o *DataGrid* e em *DataSource* escolha o *exphCategoria*. A partir daí você já pode visualizar as propriedades da classe.

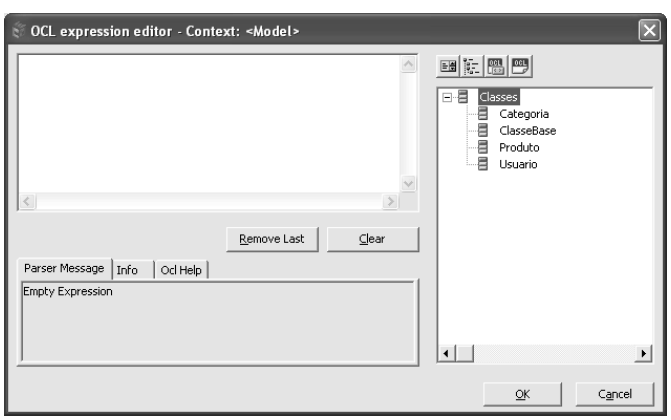


Figura 1. Editor OCL do ExpressionHandle

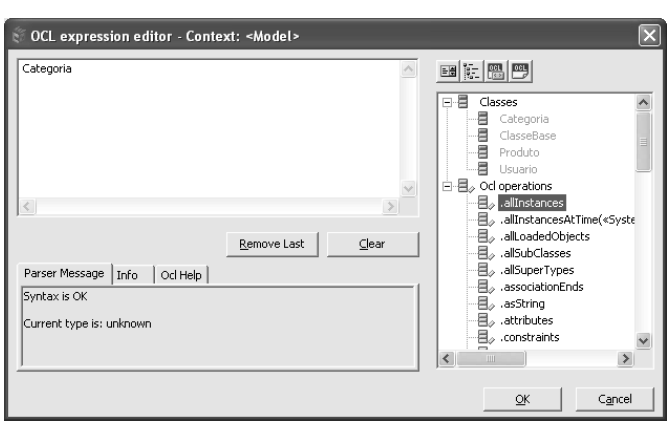


Figura 2. Lista de expressões OCL

Personalizando o DataGrid

Vamos adicionar ao *DataGrid* os botões para exclusão e edição. Clique no *DataGrid* e acesse o *Object Inspector*. Clique sobre o item *Property Builder* e na janela que surge selecione a guia *Columns*. Desmarque a opção que indica que as colunas serão geradas automaticamente.

Na lista de colunas disponíveis selecione a propriedade *Nome* e a adicione à lista de colunas selecionadas. Expanda o item *Button Column*, adicione os botões *Edit* e *Delete* e altere seu tipo para *PushButton*. Feche o editor, através do botão OK.

Adicione dois botões abaixo do *DataGrid* alterando sua propriedade *Text* para "Nova Categoria" e "Salvar", respectivamente. Nesse momento, o formulário deve parecer com o da **Figura 3**.

Fazendo funcionar

Adicione ao formulário uma referência ao *Package_1Unit* e adicione o código da **Listagem 1** ao evento *Click* do *Nova Categoria*.

Listagem 1. Criando uma nova categoria

```
procedure TWebForm1.btNovo_Click(
  sender: System.Object; e: System.EventArgs);
var
 lCategoria: Categoria;
begin
  lCategoria := Categoria.Create(EcoSpace);
  UpdateDatabase;
  DataBind;
end;
```

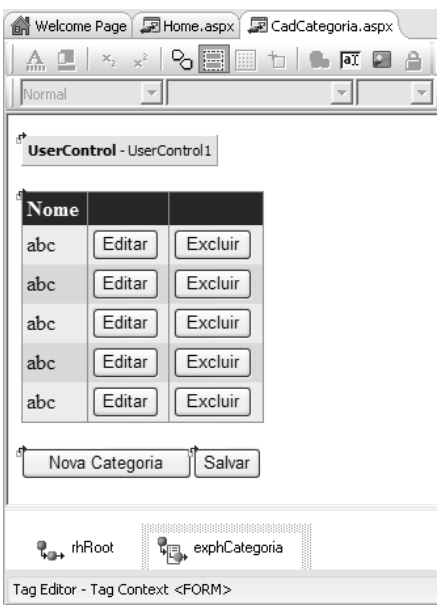


Figura 3. Layout da página CadCategoria.aspx

ClubeDelphi PLUS

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como fazer formatações no DataGrid do delphi for .NET.

www.devmedia.com.br/articles/viewcomp.asp?comp=2757

Para o *Salvar*, adicione uma chamada ao *UpdateDatabase*. Com essas duas rotinas implementadas, podemos inserir e salvar novas categorias. Falta ainda configurar o *DataGrid* para permitir a alteração e exclusão.

Acesse novamente o *Property Builder* do *DataGrid* e na guia *General*, defina que o campo chave de dados é a propriedade *ExternalID* (**Figura 4**).

Como último passo, é preciso ligar os eventos *CancelCommand*, *DeleteCommand*, *EditCommand* e *UpdateCommand* à suas respectivas implementações. Para isso acesse os eventos do *DataGrid* e configure-os conforme a **Figura 5**.

Salve o projeto e compile. Clicando com o botão direito sobre alguma área em branco do formulário e escolhendo *View in browser*. Já podemos testar o cadastro de Categorias (**Figura 6**).

Falta apenas incluir um link para o cadastro de Categorias. Abra o *User Control* e inclua uma coluna na tabela. Nessa, inclua um *Hyperlink*, ajuste sua propriedade *NavigateUrl* para "CadCategoria.aspx" e *Text* para "Categorias".

Formate a coluna e o *Hyperlink* seguindo o padrão dos outros controles. O *User Control* deve parecer com a **Figura 7**.

Cadastro de Produtos

Adicione um novo *ECO WebForm* e renomeie-o como "CadProduto.aspx". Ligue o *RootHandle* ao *ECOSpace* através do *EcoSpaceType*. Nesse momento, adicione o *User Control* e modele a página semelhante à **Figura 8**.

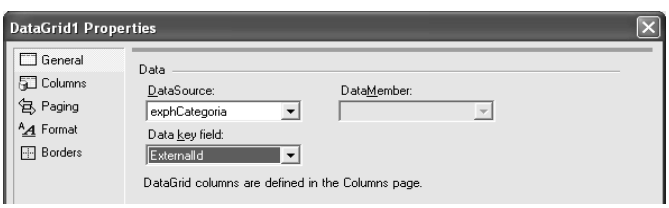


Figura 4. Definindo a chave de dados

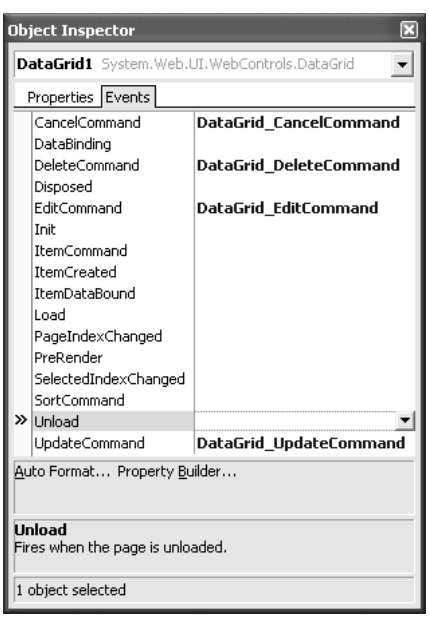


Figura 5. Ajustando os eventos do DataGrid

Vamos adicionar um *ExpressionHandle* e renomeá-lo para "exphCategoria". Ligue a propriedade *RootHandle* para o *RootHandle* presente da página e ajuste a propriedade *Expression* para "Categoria.allInstances->orderBy(Nome)".

Essa OCL trará todos os objetos do tipo *Categoria*, ordenados pela propriedade *Nome*. Altere também a propriedade *AddExternalId* para *True*, para podermos utilizar o identificador do objeto.

Evolve Database

Observe na **Figura 8** que temos um campo para o preço do produto, porém, não temos uma propriedade Preço ou Valor na classe *Produto*. Vamos adicioná-la. No *Project Manager* acesse a guia *ModelView* e dê um duplo clique sobre o *Package_1*.

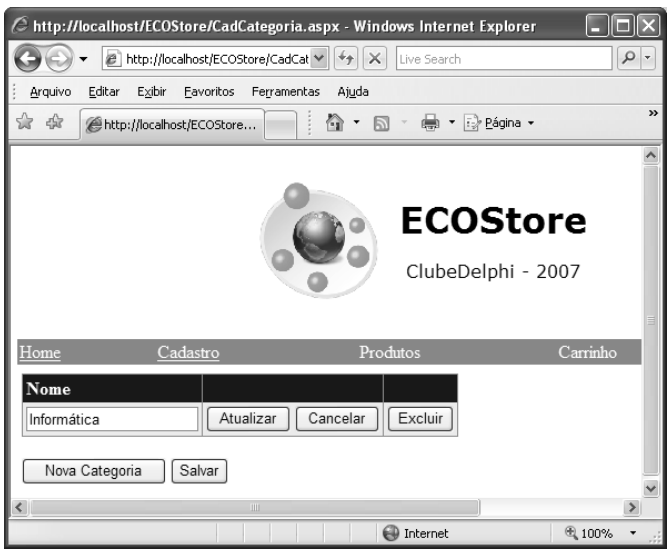


Figura 6. Cadastro de categorias em funcionamento

Figura 7. User Control ajustado para indicar as categorias

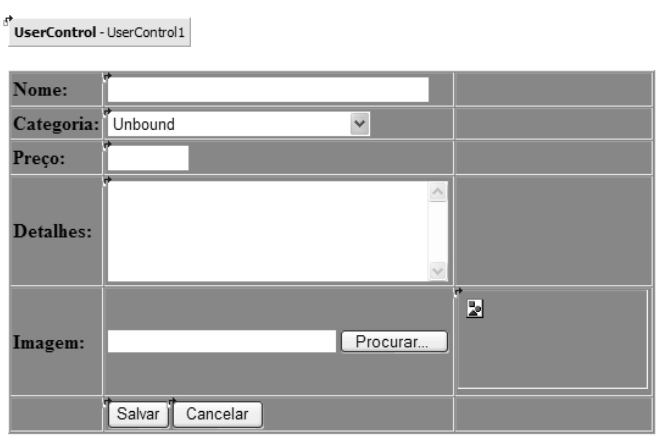


Figura 8. Layout da página CadProduto.aspx

O diagrama de classes será aberto. Edite a classe *Produto* e adicione um novo atributo chamado "Valor" do tipo *Decimal*. Feito isso, salve e compile a aplicação. Ainda no *Project Manager*, selecione a guia do projeto, abra a unit *EcoPersistenceMapperProvider.pas*.

Caso esteja sendo exibido o código da unit, mude a visualização para o modo de *design*. Clique no botão *Evolve Schema* e a janela da **Figura 9** será exibida.

Na guia *Actions* é informado que após várias análises a diferença encontrada no modelo em relação ao banco de dados é a coluna *Valor*, que será adicionada. Você pode, antes de clicar no *Execute*, selecionar a guia *SQL Script* e verificar como que será adicionada e se estiver curioso acesse as outras guias também. Clique no *Execute* para a coluna ser criada.

DataBindings complexos

Quando um controle pode exibir mais de uma instância de um determinado objeto através de *DataBinding*, dizemos que esse é complexo. Vamos ligar nossa lista de Categorias ao *DropDowList*. Selecione o controle e altere a propriedade *DataSource* para *exphCategoria*.

Na propriedade *DataTextField* definimos o que será exibido, então, abra a lista de campos disponíveis e serão listadas as propriedades da classe *Categoria*. Escolha a propriedade *Nome*.

Devemos também determinar o *DataValueField*, que representa o valor que é retornado ao selecionar um item do *DropDownList*, no nosso caso alteramos essa propriedade para apontar para o *ExternalId* de Categoria. Com isso o *DataBinding* está pronto, se executarmos agora nosso formulário, veremos a lista de Categorias.

Uma aplicação ECO oferece um recurso chamado *BindingContext* que é mantido entre o controle e a lista de objetos, funcionando como um ponteiro indicando de forma automática qual é o objeto ativo no controle.

Esse recurso está disponível em aplicações Windows Forms através do *CurrencyManagerHandle*. Para aplicações ASP.NET, não temos esse ponteiro automático e sim um *CursorHandle* que é semelhante ao *CurrencyManagerHandle* com exceção do controle automático.

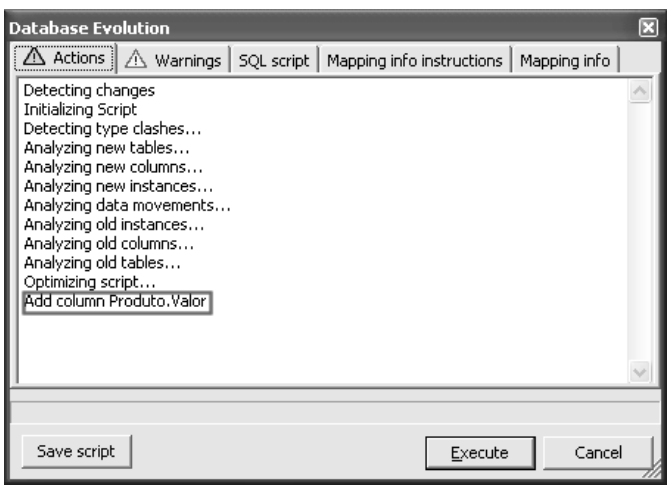


Figura 9. Janela Database Evolution

Para entender como vamos realizar os *DataBindings* é preciso explicar como nossa página de cadastro de produtos funcionará. Quando o usuário escolher o link *Produtos* um formulário de pesquisa será exibido.

Nele o usuário poderá alterar um produto já cadastrado ou cadastrar um novo. Quando o usuário escolher alterar um produto existente o Id do objeto *Produto* selecionado será passado e não sua instância.

Quando o usuário quiser criar um novo produto, o formulário de cadastro não receberá o Id do produto e criará um novo objeto. Com o *CadProduto.aspx* aberto pressione F12 para entrarmos no código. Crie na seção *private* uma variável (chamada "lProduto") do tipo *Produto*, não esqueça de adicionar ao uses o *Package_1Unit*.

Quando criamos um *ECO WebForm*, o Delphi já embute algumas rotinas pra facilitar nossa vida. Uma delas está no *Page_Load*. Vamos alterá-lo de acordo com a **Listagem 2**.

Nesse evento, determinamos que se o Id do objeto não for passado, um novo objeto é criado e inserido no *rhRoot* como elemento em uso. Caso seja passado o Id do objeto, ele é então localizado e retornado pela função *ObjectForId*, definido como objeto corrente e o *DropDowList* é então atualizado.

Listagem 2. Page_Load da CadProduto.aspx

```
procedure TWebForm1.Page_Load(sender: System.Object;
  e: System.EventArgs);

var
  Id: string;
  _item: ListItem;

begin
  EcoSpace.Active := True;
  Id := Request.Params['RootId'];

  if Assigned(Id) and (Id <> '') then
  begin
    rhRoot.SetElement(ObjectForId(Id));
    lProduto := rhRoot.Element.AsObject as Produto;
  end
  else
  begin
    lProduto := Produto.Create(EcoSpace);
    rhRoot.SetElement(lProduto);
  end;

  if not IsPostBack then
    DataBind;

  if Assigned(Id) and (Id <> '') then
  begin
    _item := ListItem.Create;
    _item := DropDownList1.Items.FindByText(
      lProduto.Categoria.Nome);
    DropDownList1.SelectedIndex := DropDownList1.Items.IndexOf(
      _item);
  end;

end;
```

Utilizaremos o *rhRoot* como fonte de dados para realização do *DataBinding* e para isso é preciso configurar que a classe o *rhRoot* vai representar. Clique nele, acesse a propriedade *StaticValueTypeName* e escolha *Produto*, que é a nossa classe.

Acesse as propriedades do *TextBox* referente ao nome do produto e clique em *DataBindings*, configurando-o conforme a **Figura 10**.

Faça o *DataBinding* para os outros controles com exceção do *DropDownList*, nele faremos manualmente devido a falta de *BindingContext* mencionado anteriormente. Para o campo *URLImagem*, use a propriedade *ImageURL* do controle *Image*.

> **Nota:** *Você pode dar uma formatação para o campo Valor, acessando o item Format e escolhendo Currency - {0:C}.*

Adicione um *CursorHandle* renomeando-o para "chCategoria". Ligue sua propriedade *RootHandle* para *exphCategoria*. O *chCategoria* fará a ponte, mesmo que manualmente, para o objeto *Categoria* que está selecionado no *DropDownList*.

Salvando o Produto

No evento *Click* do *Salvar* adicione o código da **Listagem 3** (atente para o nome dos controles que você adicionou no formulário).

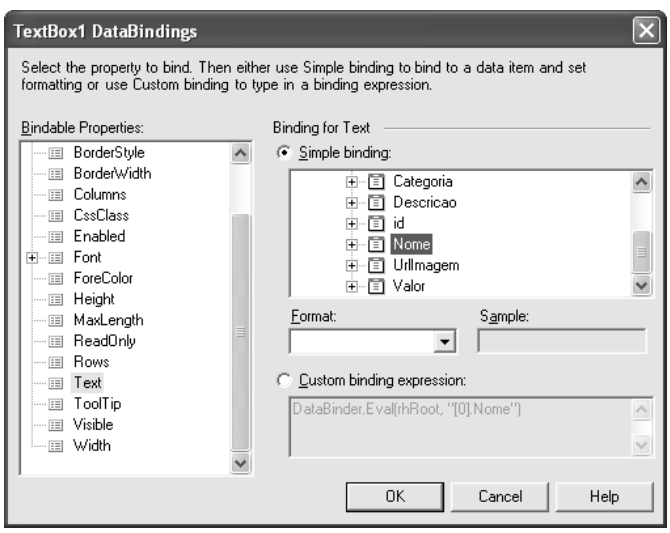


Figura 10. DataBinding no TextBox de um rhRoot

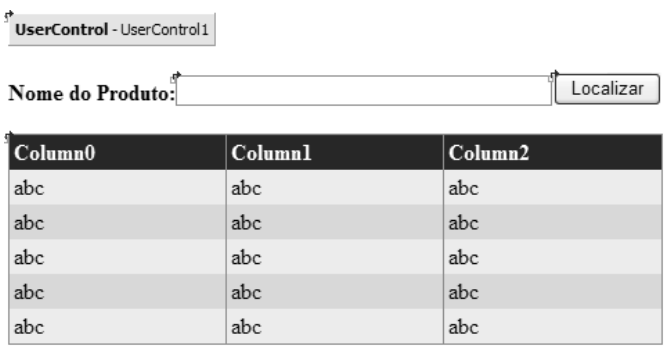


Figura 11. Formulário de consulta de Produtos

Listagem 3. Salvando um Produto

```
procedure TWebForm1.btSalvar_Click(
  sender: System.Object; e: System.EventArgs);
var
 lProduto: Produto;
begin
  lProduto := rhRoot.Element.AsObject as Produto;
  lProduto.Nome := TxtNome.Text.ToUpper;
  lProduto.Valor := Convert.ToDecimal(TxtValor.Text);
  lProduto.Descricao := TxtDetalhes.Text;
  lProduto.Categoria := ObjectForId(DropDownList1.SelectedValue).
    AsObject as Categoria;
  if SalvarImagem <> '' then
    (rhRoot.Element.AsObject as Produto).UrlImagem := SalvarImagem;
  btSalvar.Text := lProduto.Categoria.Nome;
  UpdateDatabase;
end;
```

Não descrevi o conteúdo da função *SalvarImagem*. O seu objetivo é o mesmo da *SalvarFiguraDisco* muito bem explicada na edição 75 por Luciano Pimenta, porém com algumas alterações.

Simplificando, a mesma faz uma cópia para uma pasta no servidor e salva no campo *URLImagem*, o caminho da imagem, ficando por exemplo: *http://localhost/ECOStore/imagens/figura.jpg*.

Teste o cadastro. Após o teste, vamos incluir a linha a seguir, logo após o método *UpdateDatabase* do *Salvar* e também no evento *Click* do *Cancelar*.

```
Response.Redirect('ConsultaProduto.aspx');
```

Esse código retornará para a página de consulta logo após salvarmos o objeto.

> **Nota:** *Para que possamos acessar o objeto Input File no código, clique com o botão direito no controle, e escolha Run As Server Control.*

Consulta de Produtos

Adicione um *ECO WebForm* e renomeie como "ConsultaProduto.aspx". Faça a ligação do *RootHandle* ao *ECOSpace* através da propriedade *EcoSpaceType*. Adicione o *User Control* e modele a página semelhante à **Figura 11**.

Nessa tela de consulta, o usuário fornecerá parte no nome do produto, clicará no *Localizar* e a página retornará os produtos encontrados. Caso o usuário deseje alterar alguma informação sobre o produto, bastará clicar sobre o nome do mesmo e será redirecionado para a página de cadastro.

Adicione um *ExpressionHandle*, o renomeie-o para "exphConsultaProduto", mude para *True* a propriedade *AddExternalId* e ligue-o ao *rhRoot* da página. Esse *ExpressionHandle* será utilizado em nossa consulta. Vamos ajustar o *Expression* para o seguinte OCL:

```
Produto.allInstances->select(Nome='')
```


ClubeDelphi PLUS

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como criar telas de consultas utilizando JavaScript no Delphi.

www.devmedia.com.br/articles/viewcomp.asp?comp=4270

Com esse OCL, estamos pedindo ao ECO que selecione todos os objetos do tipo Produto, onde a propriedade *Nome* esteja vazia. Dessa forma, ao executarmos o formulário, nada será trazido.

Vamos formatar o *DataGrid*. Ligue o *DataSource* ao *exphConsultaProduto*. Acesse o *Property Builder* e na guia *General* defina que campo chave de dados é o *ExternalID*. Na guia *Columns* desmarque a criação automática e adicione uma coluna do tipo *Hyperlink*.

Vamos configurá-la da seguinte forma: na propriedade *Header text* digite "Produto" e em *Text field* escolha *Nome*. Em *URL field* escolha *ExternalID* e em *URL format string* vamos chamar o formulário de cadastro de produtos, passando para ele o código interno do produto selecionado (usando uma *QueryString*).

Ajuste essa propriedade para "CadProduto.aspx?RootId={0}". Adicione um botão ao lado do *Localizar* alterando seu *Text* para "Novo Produto" e no *Click* chame a página de cadastro de produtos, usando o seguinte código:

```
Response.Redirect('CadProduto.aspx');
```

No *Click* do *Localizar* digite o seguinte código, onde estamos alterando a sentença OCL para retornar os produtos que buscamos:

```
exphConsultaProduto.Expression :=
  'Produto.allInstances->select(Nome >= ''' +
  TxtNome.Text + ''')';
DataBind;
```

Atualize o *User Control* para que ao clicar no link *Produto*, seja carregada a página de consulta dos mesmos. Execute o projeto e adicione produtos (**Figura 12**).

Conclusão

No artigo de hoje nos aprofundamos um pouco mais nos recursos do ECO, dentre eles a OCL e o *Evolve Database*, que se responsabiliza em atualizar a estrutura do banco de dados para refletir o modelo UML existente. No próximo artigo vamos criar a página principal do site, onde o usuário poderá efetuar a compra definitivamente. Não perca! ■

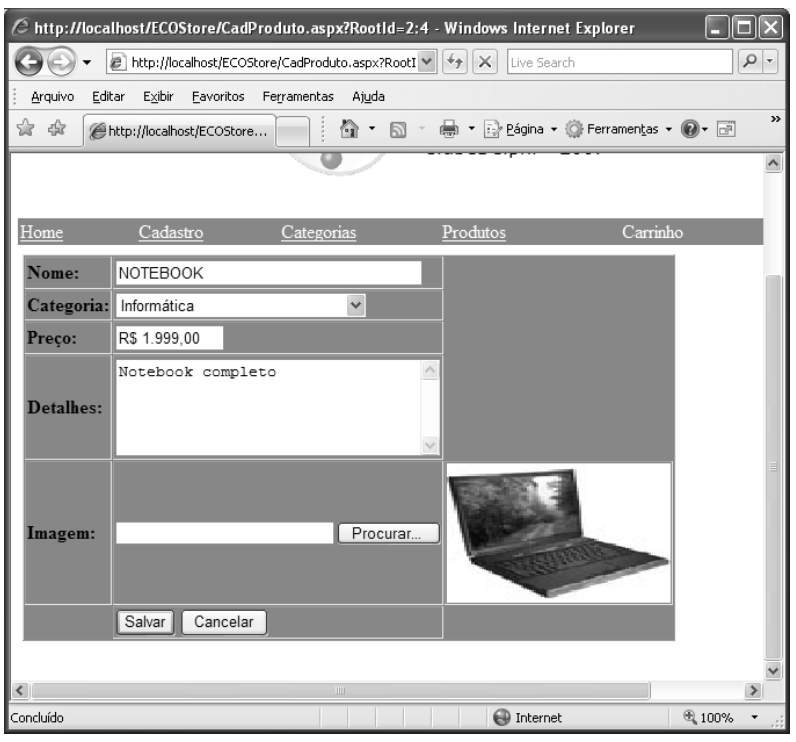


Figura 12. Testando o cadastro de Produtos

Lançamentos!

Microsoft SQL Server 2005
Fundamentos de Bancos de Dados
Bookman
R$ 68,00

Programação Orientada a Objetos Usando Delphi
4ª Edição
Visual Books
R$ 75,00

Dominando o PostgreSQL
Ciência Moderna
R$ 39,20

Qualidade de Software
2ª Edição
Ciência Moderna
R$ 179,00

Frete Grátis
Para todo o Brasil!
Até 6x sem juros no cartão.*
*Parcela mínima de R$ 20,00
Outras facilidades de pagamento:
Tel: (11) 4062-5152
livrosdeprogramacao.com.br

Validações em duas camadas

FABRÍCIO DESBESSEL
(fabricio@fabricio.pro.br)
é professor de Linguagem de Programação do Curso Técnico em Informática do Colégio Frederico Jorge Logemann de Horizontina/RS e da FAHOR Faculdade Horizontina. Delphiano de coração está sempre disposto a provar que com o Delphi sempre teremos a melhor solução. Site *www.fabricio.pro.br*.

Este artigo tem por objetivo tratar de validações em aplicações *Client-Server* (duas camadas) abordando dois cenários. O primeiro cenário tratará de todas as validações na aplicação cliente, não utilizando recursos de banco de dados. A idéia desse cenário é utilizar o banco de dados somente como um repositório de dados, abrindo a possibilidade de utilizar qualquer banco, pago ou *free*.

O segundo cenário apresentará as validações padrões, existentes em todos os bancos de dados, demonstrando como tratar essa validação na aplicação cliente, possibilitando que o usuário decida o que deve ser feito.

Para exemplificar serão utilizadas as seguintes validações:

- Validação de Chave Primária;
- Validação de Chave Estrangeira;
- Validação de Campo Nulo;
- Validação de Intervalos de Valores;
- Validação de Valores entre diferentes Campos.

Para criar esses cenários, crie um banco de dados, no Firebird ou InterBase, chamado "Empresa" e execute o *script* da **Listagem 1**.

Listagem 1. Script para criação das tabelas do banco

```
CREATE TABLE DEPARTAMENTOS
(
  CODDEPARTAMENTO
INTEGER,
  DESCRICAO           VARCHAR(60),
  SALARIO_MINIMO      NUMERIC(5,2),
  SALARIO_MAXIMO      NUMERIC(5,2)
);

CREATE TABLE EMPREGADOS
(
  CODEMPREGADO        INTEGER,
  CODDEPARTAMENTO     INTEGER,
  NOMECOMPLETO        VARCHAR(100),
  CPF                 VARCHAR(11),
  TELEFONE            VARCHAR(20)
);
```

Note que no *script* não temos campos *Not Null*, nem *Primary Key* e *Foreign Key*. Agora vamos começar nossa aplicação que,

depois de pronta apresentará os dois cenários. Crie uma nova aplicação no Delphi e também um Data Module com o nome de "DMDados", que será o repositório dos componentes de conexão ao banco.

Os componentes de acesso e suas respectivas configurações são:

- SQLConnection
  - *Name*: "sqlConexao"
  - *ConnectionName*: IBConnection
  - *LoginPrompt*: *False*
  - *DriverName: Interbase*
  - *Params[SQLDialect]*: 3
  - *Params[Database]*: <caminho do banco EMPRESA.FDB>
- SQLDataSet
  - *Name*: "sdsEmpregado"
  - *SQLConnection*: *sqlConexao*
  - *CommandType*: *ctTable*
  - *CommandText*: EMPREGADOS
- DataSetProvider
  - *Name*: "dspEmpregados"
  - *DataSet*: *sdsEmpregados*
  - *UpdateMode: upWhereKeyOnly*
- ClientDataSet
  - *Name*: "cdsEmpregados"
  - *ProviderName*: *dspEmpregados*
- DataSource
  - *Name*: "dsEmpregados"
  - *DataSet*: *cdsEmpregados*

Obs.: *Repita esse processo (com exceção do SQLConnection) para a tabela DEPARTAMENTOS.*

Crie um formulário para edição dos dados da tabela *Empregados*. Pode ser um formulário bem simples, com a utilização do *DBNavigator* e um botão com o comando *ApplyUpdate(0)*.

1º Cenário: Chave primária

Podemos, de forma muito elegante, definir um campo chave diretamente na aplicação, mesmo ela não existindo no banco de dados. Para tanto é necessário dar um duplo clique no *sdsEmpregados* e adicionar os campos (*TFields*).

Depois, selecione o campo *CodEmpregado* (que será nossa chave primária) e nas sub-propriedades de *ProviderFlags* deixamos como *True* a *pfInUpdate*, *pfInWhere* e *pfInKey*. Essa última propriedade é que estipula o campo chave.

As outras informam como o *DataSetProvider* deverá gerar as instruções de *Insert* e *Update*. Perceba que informamos que o campo *CodEmpregado* deverá estar presente nas cláusulas *Update* e *Where* da instrução SQL.

Agora selecione os campos restantes e deixe somente *pfInUpdate* com *True*. Veja na **Figura 1** a exceção de chave primária em execução.

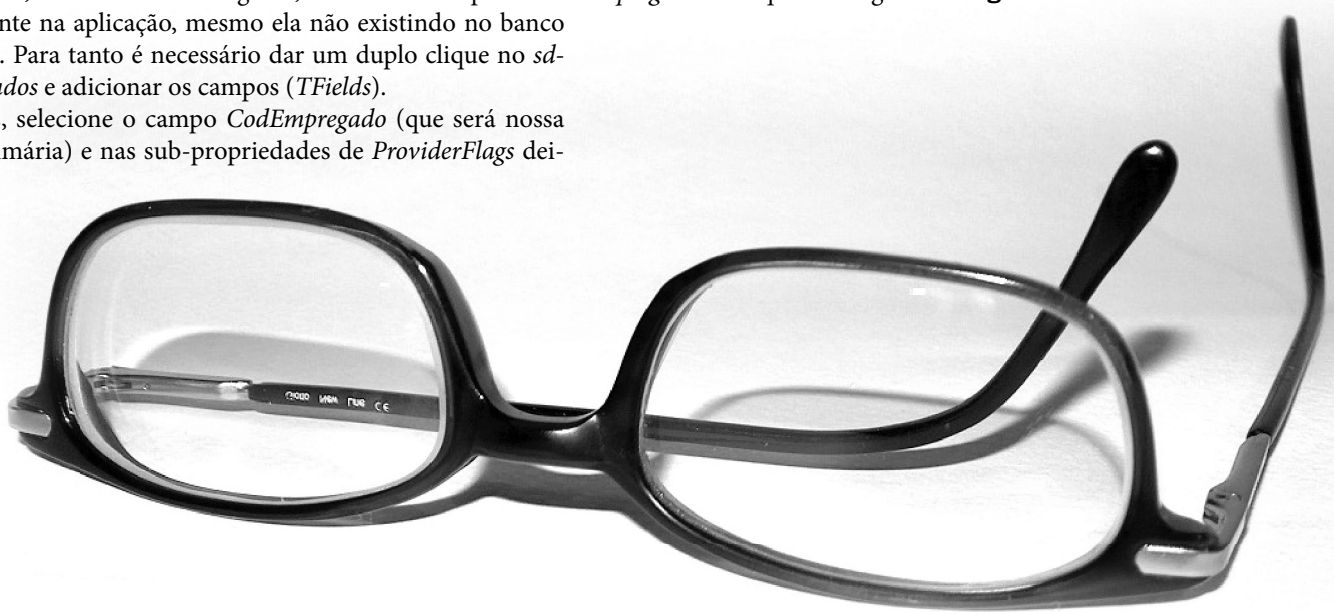
Chave estrangeira

Para tratarmos dessa validação, será necessário utilizar o evento *BeforeUpdate* do *DataSetProvider*. Esse evento acontece antes da aplicação enviar os dados para o *Update* no banco de dados.

Na nossa aplicação exemplo, verificaremos se o código de departamento informado no cadastro de empregados corresponde a um departamento existente. Vamos adicionar mais três componentes para fazer essa consulta, conforme as configurações a seguir:

- SQLDataSet
  - *Name*: "sdsChave"
  - *SQLConnection*: *sqlConexao*
  - *CommandType*: *ctQuery*
  - *CommandText*: "select CODEMPREGADO
    from EMPREGADOS
    where CODEMPREGADO = :CODIGO"
- DataSetProvider
  - *Name*: "dspChave"
  - *DataSet*: sdsChave
  - *Options>poAllowCommandText*: *True*
- ClientDataSet
  - *Name*: "cdsChave"
  - *ProviderName*: *dspChave*

Alteramos a propriedade *poAllowCommandText* do *DataSetProvider* para *True* possibilitando a alteração do comando SQL no *ClientDataSet*. No evento *BeforeUpdateRecord* do *dspEmpregados* coloque o código da **Listagem 2.**



Listagem 2. Código do evento BeforeUpdateRecord

```
procedure TDados.dspEmpregadosBeforeUpdateRecord(
  Sender: TObject; SourceDS: TDataSet;
  DeltaDS: TCustomClientDataSet;
  UpdateKind: TUpdateKind; var Applied: Boolean);

begin
  if UpdateKind in [ukModify, ukInsert] then
  begin
    { validação de Chave Estrangeira }
    cdsChave.Close;
    cdsChave.CommandText := 'select CODDEPARTAMENTO '+
      'from DEPARTAMENTOS '+
      'where CODDEPARTAMENTO = :Codigo';
    cdsChave.Params.ParamValues['Codigo']:=
      DeltaDS.FieldByName('CODDEPARTAMENTO').NewValue;
    cdsChave.Open;

    if cdsChave.RecordCount = 0 then
      raise Exception.Create('Erro de Chave Estrangeira !'+#13+
        'Registro não existente');
  end;

end;
```

Explicando: Utilizamos uma consulta SQL para verificar se existe um departamento com o código informado. Para pegar o código informado nesse evento, utilizamos o *DeltaDS*, que traz os campos e informações que serão alterados ou inseridos no banco.

Inclusive com o *DeltaDS* podemos pegar as novos valores (*NewValue*) e os valores antigos (*OldValue*). Na **Figura 2** podemos ver a exceção levantada para impedir que o registro seja cadastrado, estando com um código de departamento inexistente.

Campo Nulo

Para determinar campos que não podem ser nulos, ou seja, devem ser informados, colocaremos a propriedade *Required* do *TField* como *True*. Então no *sdsEmpregados*, de um duplo clique, selecione os campos que deverão ser preenchidos pelos usuários e mude a propriedade *Required* para *True* (faça o mesmo no *cdsEmpregados*). Na **Figura 3** podemos ver a exceção.

Intervalo de Valores

No mesmo sentido, os campos *TField* também possuem uma propriedade chamada *CustomConstraint* para colocar regras específicas do campo. Para testar, selecione o campo Salário Mínimo do *sdsDepartamentos* e informe a seguinte regra: "X > 100 AND X < 400".

Na propriedade *ConstraintErrorMessage* você pode colocar a mensagem que será exibida. Para o exemplo coloquei a seguinte: "Valor fora da Faixa. (100-400)". O interessante é que podemos utilizar quaisquer regras SQL nessa propriedade, possibilitando inúmeras validações.

Veja na **Figura 4** a exceção gerada (crie um cadastro simples para a tabela DEPARTAMENTOS).

Valores entre diferentes campos

Assim como um campo pode ter *constraints*, a tabela inteira também pode ter. Com isso temos a possibilidade de validar valores testando campos diferentes. Isso pode ser feito através da propriedade *Constraints* do *ClientDataSet*.

Para testar, vamos especificar que o Salário Máximo deva ser maior que o Salário Mínimo. Então, no *cdsDepartamentos* clique na propriedade *Constraints*, abrindo o editor de *constraints*, onde pode-se inserir várias regras.

Listagem 3. Script do banco com algumas validações

```
CREATE TABLE DEPARTAMENTOS
(
  CODDEPARTAMENTO  INTEGER NOT NULL PRIMARY KEY,
  DESCRICAO        VARCHAR(60) NOT NULL,
  SALARIO_MINIMO   NUMERIC(5,2),
  SALARIO_MAXIMO   NUMERIC(5,2)
);

CREATE TABLE EMPREGADOS
(
  CODEMPREGADO     INTEGER NOT NULL PRIMARY KEY,
  CODDEPARTAMENTO  INTEGER NOT NULL,
  NOMECOMPLETO     VARCHAR(100) NOT NULL,
  CPF              VARCHAR(11),
  TELEFONE         VARCHAR(20),
  CONSTRAINT FK_DEPTO
  FOREIGN KEY ("CODDEPARTAMENTO")
  REFERENCES DEPARTAMENTOS (CODDEPARTAMENTO)
);
```

Listagem 4. Evento OnReconcileError

```
procedure TDataModule2.cdsEmpregadosReconcileError(
  DataSet: TCustomClientDataSet; E: EReconcileError;
  UpdateKind: TUpdateKind;
  var Action: TReconcileAction);
begin
  Action := HandleReconcileError(DataSet, UpdateKind, E);
end;
```

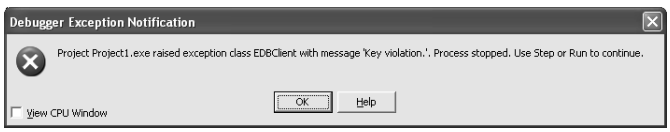


Figura 1. Exceção de chave primária

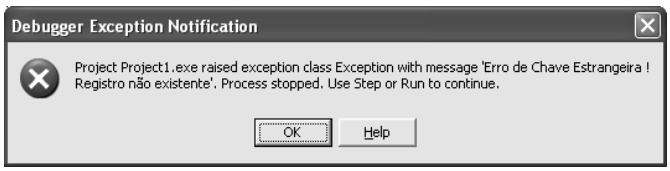


Figura 2. Validação de chave estrangeira

Figura 3. Validação de campo nulo

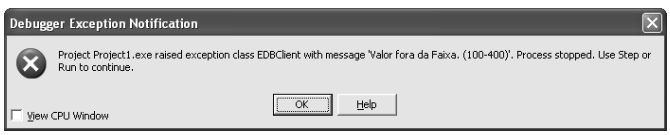


Figura 4. Exceção de faixa de valores

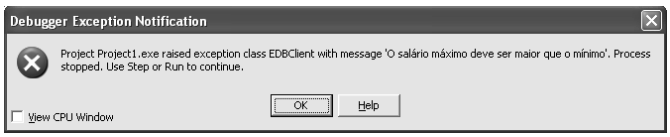


Figura 5. Exceção entre diferentes campos

Clique no botão *New* para criar uma regra e na propriedade *CustomConstraint* informe: "SALARIO_MAXIMO > SALARIO_MINIMO". Na propriedade *ErrorMessage* digite: "O salário máximo deve ser maior que o mínimo". Com isso teremos uma validação entre campos, conforme mostra a **Figura 5**.

2º Cenário

Para o segundo cenário, vamos mostrar as três primeiras validações (chave primária, chave estrangeira e campo nulo). Vamos criar um novo banco de dados que contenha essas validações.

> **Dica:** *Na edição 77 tem um artigo sobre validações em banco de dados, partindo de um problema e sua solução, através de regras.*

Crie um novo banco de dados (se desejar, crie apenas as tabelas com nomes diferentes das já criadas) e execute o *script* da **Listagem 3**.

Como o objetivo desse artigo é facilitar o máximo, utilizaremos um formulário padrão do Delphi, chamado de *ReconcileError*. Esse formulário é utilizado para tratar todas as exceções que acontecem no banco de dados, possibilitando que nossa aplicação interfira no processo.

Inclusive possibilita que o usuário altere os dados, ou simplesmente desista da alteração ou inserção. Crie uma nova aplicação e repita os passos necessários para inclusão dos componentes de acesso a dados. Crie um formulário de edição de dados para a tabela *Empregados*.

Para utilizar o *ReconcileError*, acesse o menu *File>New>Other* e escolha a aba *Dialogs*. Selecione o item *Reconcile Error Dialog*. Agora é necessário chamar o formulário. Para tanto, no evento *OnReconcileError* do *cdsEmpregados* (*ClientDataSet*) coloque o código exibido na **Listagem 4**.

Adicione a *unit* do *ReconcileError* no Data Module. Isso pode ser feito através do menu *File>Use unit*. Para testar o funcionamento, adicionei um departamento de código 1 na tabela de departamentos e tentei fazer uma inclusão de empregado informando o código do departamento 5, o que deve resultar e um erro de chave estrangeira. Veja a **Figura 6** que mostra o *ReconcileError* em ação.

Quando o *ReconcileError* é chamado apresentará os dados que foram informados na aplicação, inclusive oferecendo a opção de visualizar somente os campos que sofreram alteração. Além disso, nos possibilita três ações para tratar a exceção (*Reconcile Action*):

- *Skip*: Simplesmente "escapa" sem tratar o evento, voltando para o formulário que o usuário estava;
- *Cancel*: Cancela a inserção ou alteração do registro;
- *Correct*: A mais interessante de todas, pois ao selecioná-la é dada a possibilidade ao usuário da alteração dos dados no próprio formulário.

Além dessas três ações mais comuns, temos outras ações para tratar conflitos de registros que foram alterados ao mesmo tempo por mais usuários:

- *Abort*: Aborta o tratamento voltando para o formulário que o usuário estava;
- *Merge*: Possibilita a verificação dos dados que foram alterados por outros usuários e a junção com os dados alterados na sua aplicação.

Na aplicação teste, escolhi a opção *Correct* e informei 1 para o campo CODDEPARTAMENTO. Mesmo assim, ainda é necessário executar um novo *Apply* no banco de dados para que as alterações realizadas no *ReconcileError* sejam efetivamente salvas.

Agora, para você utilizar o *ReconcileError* em suas aplicações, falta somente fazer algumas traduções para que fique mais entendível para o usuário. Todos os componentes visíveis podem ser facilmente traduzidos.

O trabalho mais pesado fica na hora de fazer algumas traduções nos código. Comece alterando as constantes conforme mostra a **Listagem 5**.

Outra possibilidade interessante é alterar o evento *FormCreate* do *ReconcileError* colocando alguns testes para verificar o tipo de erro que ocorreu e acrescentar uma tradução ao erro. Veja como deve ficar o código desse evento na **Listagem 6**.

Depois dessas alterações teremos todos os erros de validação devidamente tratados, através de uma apresentação ao usuário, que poderá decidir o que fazer. Veja na **Figura 7** o *ReconcileError* com as traduções.

Conclusão

É importante salientar que o primeiro cenário apresentou todas as validações na aplicação, sem utilizar quaisquer características do banco de dados escolhido. É uma boa opção para quem quer realmente ter um sistema multi banco de dados.

Se essa for a escolha, sugiro que sejam criados índices para cada tabela, permitindo que os registros sejam encontrados mais rapidamente. Como não definimos campos chave, as tabelas estão totalmente desprovidas de índices, o que não é bom para a performance do banco de dados.

Outro ponto a salientar é que se o primeiro cenário foi o escolhido, investimentos em banco de dados de alto custo não serão vantajosos, uma vez que qualquer banco de dados pode guardar informação.

No segundo cenário vimos uma forma muito elegante de apresentar os erros ao usuário, permitindo que o mesmo decida que ações deve tomar. Agora só falta você colocar a mão na massa! ■

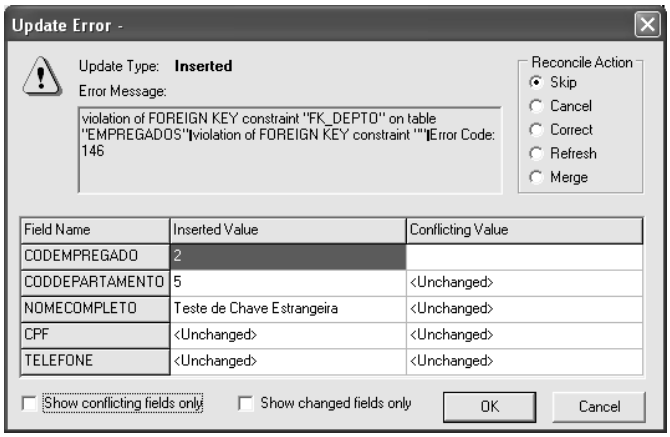


Figura 6. ReconcileError em ação

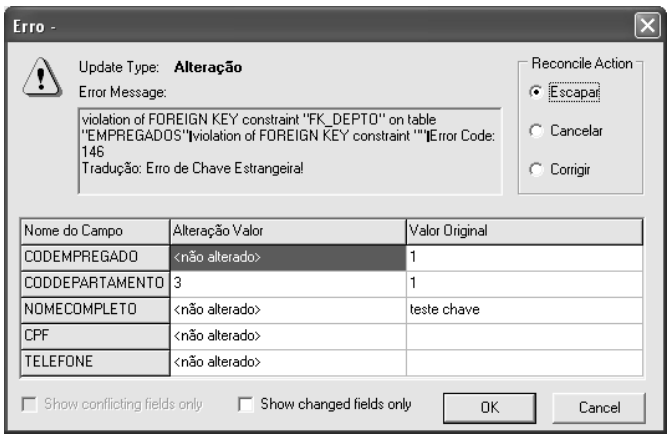


Figura 7. ReconcileError traduzido

Listagem 5. Tradução das constantes

```
const
  ActionStr: array[TReconcileAction] of string =
    ('Escapar', 'Abortar', 'Merge', 'Corrigir',
    'Cancelar', 'Atualizar');
  UpdateKindStr: array[TUpdateKind] of string =
    ('Alteração', 'Inserção', 'Exclusão');
  SCaption = 'Erro - %s';
  SUnchanged = '<não alterado>';
  SBinary = '(Binary)';
  SAdt = '(ADT)';
  SArray = '(Array)';
  SFieldName = 'Nome do Campo';
  SOriginal = 'Valor Original';
  SConflict = 'Valor conflitante';
  SValue = ' Valor';
  SNoData = '<sem registro>';
  SNew = 'Novo';
```

Listagem 6. Evento FormCreate traduzido

```
procedure TReconcileErrorForm.FormCreate(
  Sender: TObject);
begin
  if FDataSet = nil then Exit;
  FDataFields := TList.Create;
  InitDataFields;
  Caption := Format(SCaption, [FDataSet.Name]);
  UpdateType.Caption := UpdateKindStr[FUpdateKind];
  ErrorMsg.Text := FError.Message;
  if FError.Context <> '' then
    ErrorMsg.Lines.Add(FError.Context);
  if Pos('PRIMARY',ErrorMsg.Text) > 0 then
    ErrorMsg.Lines.Add(
    'Tradução: Erro de Chave Primária!');
  if Pos('FOREIGN',ErrorMsg.Text) > 0 then
    ErrorMsg.Lines.Add(
    'Tradução: Erro de Chave Estrangeira!');
  if Pos('*** null ***',ErrorMsg.Text) > 0 then
    ErrorMsg.Lines.Add(
    'Tradução: Campo obrigatório não foi informado!');
  ConflictsOnly.Enabled := FCurColIdx > 0;
  ConflictsOnly.Checked := ConflictsOnly.Enabled;
  ChangedOnly.Enabled := FNewColIdx > 0;
  InitReconcileActions;
  UpdateData.DefaultRowHeight :=
    UpdateData.Canvas.TextHeight('SWgjp') + 7;
end;
```

A EDIÇÃO QUE VOCÊ PRECISA ESTÁ ESGOTADA?

Seja um assinante Gold!

Com a assinatura Gold você já pode consultar online
todos os artigos publicados na sua revista desde a edição nº 1.

DevMedia GROUP

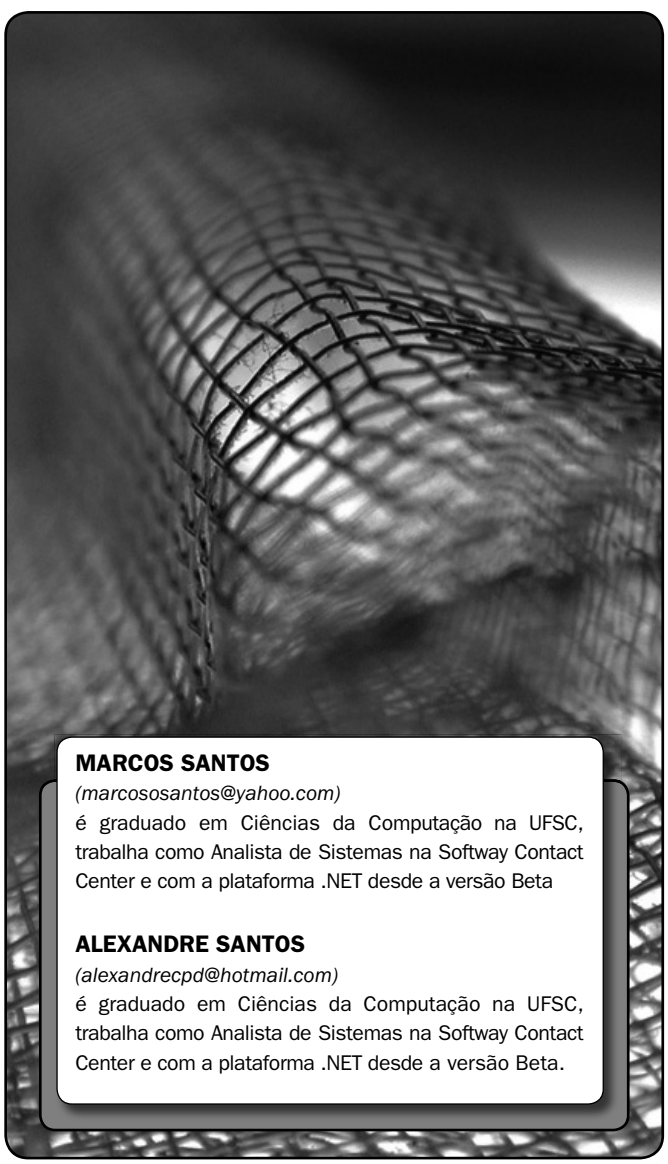
Saiba Mais! Acesse:
www.devmedia.com.br/assgold

Para mais informações:
www.devmedia.com.br/central

Assinatura Gold

Atenção: Já encontram-se disponíveis as assinaturas GOLD das revistas WebMobile e .net Magazine.

DataGrid - Criando colunas dinamicamente



MARCOS SANTOS
(marcososantos@yahoo.com)
é graduado em Ciências da Computação na UFSC, trabalha como Analista de Sistemas na Softway Contact Center e com a plataforma .NET desde a versão Beta

ALEXANDRE SANTOS
(alexandrecpd@hotmail.com)
é graduado em Ciências da Computação na UFSC, trabalha como Analista de Sistemas na Softway Contact Center e com a plataforma .NET desde a versão Beta.

Desde o lançamento do ASP.NET, o *DataGrid* é um dos controles mais empregados, principalmente pela sua fácil utilização e pela quantidade de recursos que auxiliam o desenvolvedor a criar páginas robustas e atrativas.

Com poucos cliques, tem-se acesso aos *wizards* que configuram as colunas (*Property Builder*) e o formato visual (*Auto Format*), restando apenas atribuir o *DataSource*, podendo ser um *DataSet*, *DataTable*, coleções, entre outros.

O objetivo deste artigo é mostrar a possibilidade de formatação e criação de colunas no *DataGrid* de forma dinâmica. É criado um projeto ASP.NET que efetua uma consulta de *Produtos* com suas respectivas *Categorias*, em um banco de dados do Firebird.

> **Nota:** *O banco de dados que acompanha o Firebird (Employee) não possui essas tabelas, então foi disponibilizado para download um banco com as respectivas tabelas.*

Deseja-se apresentar ao usuário todos os *Produtos* agrupados por *Categorias*, com subtotais. Para isso, é utilizado como *DataSource* um *DataTable* criado a partir do retorno da consulta. Talvez você esteja se perguntando: "Por que criar colunas dinamicamente, se é possível criá-las sem uma linha de código e de forma quase que instantânea?". Certo?

A resposta para essa pergunta pode ser melhor analisada através do seguinte questionamento: "Como fazer para que um mesmo *DataGrid* seja parametrizado de forma que em certas situações apresente X colunas e em outras situações apresente Y colunas?". Resposta: Criar as colunas dinamicamente, podendo também utilizar artifícios de Orientação a Objetos como Herança e Polimorfismo.

Para aumentar a compreensão da resposta apresentada, imagine que você queira centralizar seus relatórios em apenas uma

página ASPX com apenas um *DataGrid* e que cada relatório tenha uma consulta SQL diferente. Uma forma viável, através de OO, é criar uma classe pai *RelatorioPadrao* com dois métodos: *retornaColunas* e *retornaDados* e para cada relatório a ser gerado, criar classes filhas (exemplo *RelatorioProduto*) que irão sobrescrever esses métodos.

Dessa forma, na página ASPX será instanciada a classe desejada e invocados os métodos que retornam as colunas do *DataGrid* e seu *DataSource*. Neste artigo, para simplificar, não será modelado através de classes e subclasses.

A criação das colunas, formatação do *DataGrid* e agrupamento com subtotais serão feitos diretamente na classe da interface (ASPX).

Dica: *Para uma introdução ao uso do DataGrid no Delphi, veja o artigo de Rodrigo Sendin na edição 73.*

Projeto

No Delphi 8, 2005 ou 2006 crie um projeto ASP.NET chamado "DataGridAgrupadoColunas" contendo um formulário chamado "GridAgrupadoColunas.aspx". Adicione um *DataGrid* ("gridProdutoCategorias"), um *Button* ("btnSelecionados") e um *ListBox* ("lstSelecionados").

Como sugestão de *layout*, veja a **Figura 1**. Apenas observe que são utilizadas tabelas HTML para possibilitar uma melhor organização visual.

Para agilizar o tempo de desenvolvimento, selecione o *DataGrid* e formate a cor e fonte do cabeçalho. Em tempo de *design* não é necessário fazer mais nada. No projeto sugerido, é realizada uma consulta ao banco de dados *Employee* do Firebird, retornando a tabela *Produtos* com sua respectiva *Categorias*, como na **Figura 2**.

Porém, o desejado é que os produtos sejam visualizados agrupados por categorias no *DataGrid*, incluindo subtotais e no final um Total Geral. Para isso, via programação, a partir da consulta original, é criado um novo *DataTable* no modelo final.

No *DataGrid* são criadas cinco colunas dinamicamente, como vemos na **Tabela 1**.

Nome	Tipo	Obs
Selecionar	TemplateColumn	CheckBox
Descrição	BoundColumn	
Valor Unitário	BoundColumn	
Quantidade Estoque	BoundColumn	
Valor Estoque	BoundColumn	

Tabela 1. Tipos de colunas a serem criadas automaticamente

BoundColumns são colunas que se relacionam com os campos do *DataSource* do *DataGrid* em questão. *TemplateColumns* são colunas complexas, que permitem a inserção de diversos outros controles, como por exemplo um *CheckBox*.

Nesse caso, a coluna *Selecionar* possuirá um *CheckBox* em todas as linhas, menos nos grupos e subtotais, permitindo que o usuário informe que produtos deseja selecionar. E quando esse clicar no *Mostrar Selecionados*, os produtos em questão serão adicionados ao *ListBox*.

Essa funcionalidade é para explicitar que mesmo sendo adicionados controles em *run time*, seus valores permanecem entre *postbacks*. Precisamos então partir para o código!

ClubeDelphi PLUS

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como criar um master/detail utilizando o DataGrid.

www.devmedia.com.br/articles/viewcomp.asp?comp=2050

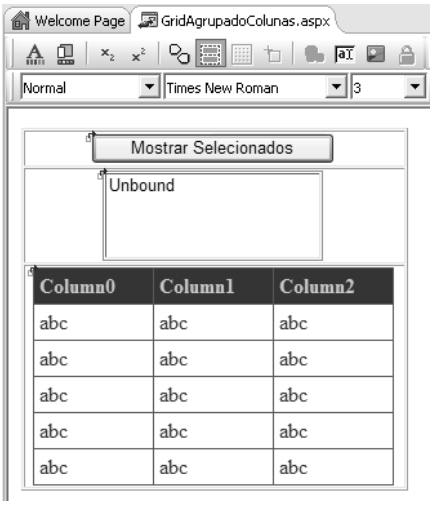


Figura 1. Layout de interface sugerido

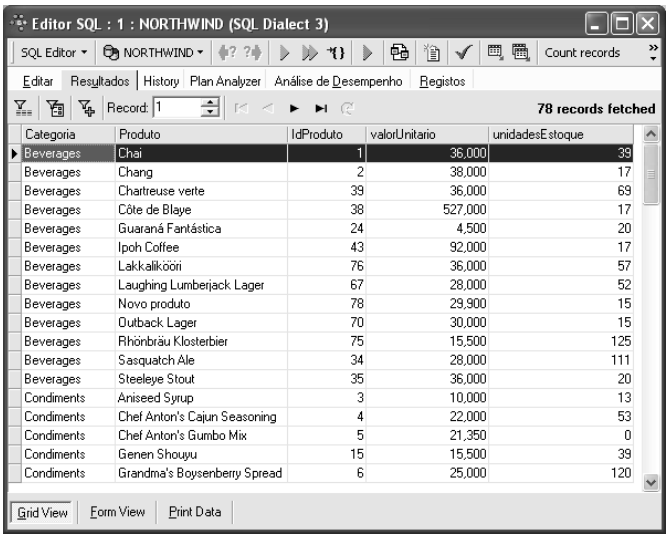


Figura 2. Consulta realizada nos Produtos e Categorias

Código

A primeira parte a ser desenvolvida é a criação das colunas do *DataGrid*, para isso, no *OnInit* deve ser invocada o *criaColunasGrid*. A necessidade da chamada do método, ser no *OnInit* (**Listagem 1**) é que nesse evento, os controles ainda não são preenchidos com seus dados pelo *DataView*.

Digite o *criaColunasGrid* da **Listagem 2**, que efetivamente cria as colunas no *DataGrid*.

Perceba que para as colunas *BoundColumn* é necessário informar o campo referente ao seu *DataSource*, através da propriedade *DataField*. Possibilita-se também a formatação através de outras propriedades, como *HeaderText* (nome no cabeçalho), *DataFormatString*, *HorizontalAlign*, entre outras.

Para as colunas *TemplateColumn*, a propriedade *ItemTemplate* permite indicar que controles serão adicionados e visualizados. Isso é possível indicando um objeto de uma classe que implementa a interface *ITemplate*, que no nosso exemplo será um *CheckBox*.

Para isso, crie uma nova classe (adicione uma nova unit) chamada "TemplateColumnCheckBox", conforme vemos na **Listagem 3**.

Até aqui, as colunas do *DataGrid* já estão sendo criadas dinamicamente. O próximo passo é realizar a consulta ao banco de dados, que será manipulada para gerar o *DataTable* no formato final, que servirá de *DataSource* ao *DataGrid*.

Para isso, digite o método *retornaCategoriasProdutos* conforme a **Listagem 4** (volte a página principal).

> **Nota:** *Usaremos o Provider do Firebird para conectar ao banco de dados. Para saber mais sobre a instalação e uso desse, veja as edições anteriores da ClubeDelphi ou várias vídeo aulas sobre o assunto no site.*

Cabe ressaltar que a conexão deve ser configurada de acordo com o local do seu banco de dados. Não esqueça de adicionar o namespace *FirebirdSql.Data.Firebird* no *uses*. Antes de criar o *DataTable* com os dados agrupados, é necessário declarar alguns atributos na seção *private* (**Listagem 5**), que serão utilizados para controle de quebra de categoria, totalizadores e o próprio *DataTable* (*dtDadosAgrupados*).

O código que instancia e cria as colunas no *DataTable* conterá os dados agrupados, pelo *criaDataTableDadosAgrupados*

Listagem 1. OnInit da página

```
procedure TWebForm1.OnInit(e: EventArgs);
begin
  InitializeComponent;
  inherited OnInit(e);
  { Dever ser invocado aqui para manter a renderização
    das colunas, através de postbacks }
  criaColunasGrid;
end;
```

Listagem 2. Método criaColunasGrid

```
procedure TWebForm1.criaColunasGrid;
var
  colSelecionar: TemplateColumn;
  colDescricao, colValorUnitario, colUnidadesEstoque,
  colValorEstoque: BoundColumn;
begin
{ Colunas não geradas automaticamente }
gridProdutoCategorias.AutoGenerateColumns = False;

{ Coluna Template que nos permite adicionar o checkbox }
colSelecionar := TemplateColumn.Create;
colSelecionar.HeaderText := 'Selecionar';

{ Informa o que o ItemTemplate é do tipo TemplateColumnCheckBox }
colSelecionar.ItemTemplate := TemplateColumnCheckBox.Create();

{ Coluna Descrição }
colDescricao := BoundColumn.Create;
colDescricao.DataField := 'Descricao';
colDescricao.HeaderText := 'Descrição';
colDescricao.ItemStyle.HorizontalAlign :=
   System.Web.UI.WebControls.HorizontalAlign.Left;

{ Coluna Valor Unitário }
colValorUnitario := BoundColumn.Create;
colValorUnitario.DataField := 'valorUnitario';
colValorUnitario.HeaderText := 'Valor Unitário';
colValorUnitario.DataFormatString  := '{0:F2}';
colValorUnitario.ItemStyle.HorizontalAlign :=
   System.Web.UI.WebControls.HorizontalAlign.Right;

{ Coluna Unidades no Estoque }
colUnidadesEstoque := BoundColumn.Create;
colUnidadesEstoque.DataField := 'unidadesEstoque';
colUnidadesEstoque.HeaderText := 'Quantidade Estoque';
colUnidadesEstoque.ItemStyle.HorizontalAlign :=
   System.Web.UI.WebControls.HorizontalAlign.Right;

{ Coluna (ValorUnitario * UnidadesEstoque) }
colValorEstoque := BoundColumn.Create;
colValorEstoque.DataField := 'valorEstoque';
colValorEstoque.HeaderText := 'Valor Estoque';
 { Duas casas decimais }
colValorEstoque.DataFormatString := '{0:F2}';
colValorEstoque.ItemStyle.HorizontalAlign :=
   System.Web.UI.WebControls.HorizontalAlign.Right;

{ Adiciona as colunas criadas no DataGrid }
gridProdutoCategorias.Columns.Add(colSelecionar);
  gridProdutoCategorias.Columns.Add(colDescricao);
gridProdutoCategorias.Columns.Add(colValorUnitario);
gridProdutoCategorias.Columns.Add(colUnidadesEstoque);
gridProdutoCategorias.Columns.Add(colValorEstoque);
end;
```

Listagem 3. Classe TemplateColumnCheckBox

```
unit uTemplateColumnCheckBox;
interface
uses
  System.Web.UI.WebControls, System.Web.UI;

type
  TemplateColumnCheckBox = class(&object, System.Web.UI.ITemplate)
  public
    procedure InstantiateIn(container: System.Web.UI.Control);
  end;

implementation

{ TemplateColumnCheckBox }

procedure TemplateColumnCheckBox.InstantiateIn(
  container: System.Web.UI.Control);
var
  chRetorno: CheckBox;
begin
 { Adiciona um checkbox na coluna template }
chRetorno := CheckBox.Create;
chRetorno.ID := 'chSelecionado';
container.Controls.Add(chRetorno);
end;
end.
```

Listagem 4. Método retornaCategoriasProdutos

```
{ Retorna o DataTable que será navegado, para gerar o
  DataTable agrupado (no formato final) }
uses System.Text, uTemplateColumnCheckBox;
...
function TWebForm1.
  retornaCategoriasProdutos: DataTable;

var
  sConexao: string;
  sSql: StringBuilder;
  dsConsulta: DataSet;
  conexao: FbConnection;
  daConsulta: FbDataAdapter;

begin
  { Faz a conexão com o banco de dados e realiza a
    consulta com os dados primários que serão
    manipulados para servir de fonte de dados ao DataGrid }
 sConexao := 'User=SYSDBA;Password=masterkey;'+
    'Database=C:\NORTHWIND.FDB;DataSource=localhost;'+
    'Port=3050;Dialect=3;Charset=NONE;Role=;'+
    'Connection lifetime=0;Connection timeout=15;'+
    'Pooling=True;PacketSize=8192;Server Type=0';
 sSql := StringBuilder.Create;;
 sSql.Append(' select c.CategoryName as Categoria, '+
    'p.ProductName as Produto, '+ 'p.ProductId as IdProduto, ');
 sSql.Append(' UnitPrice as valorUnitario, ');
 sSql.Append(' UnitsInStock as unidadesEstoque ');
 sSql.Append(' from categories c join products p ');
 sSql.Append(' on c.categoryid = p.categoryid ');
 sSql.Append(' order by c.CategoryName, '+ 'p.ProductName');
 conexao := FbConnection.Create(sConexao);
 conexao.Open();
 daConsulta := FbDataAdapter.Create(sSql.ToString, conexao);
 dsConsulta := DataSet.Create;
 daConsulta.Fill(dsConsulta);
  Result := dsConsulta.Tables[0];
end;
```

Listagem 5. Atributos a serem usados

```
type
  TipoLinha = (DescricaoCategoria, DescricaoProduto,
    SubTotal, Total);
  ...
  private
    { Variável que controla a troca de categoria }
    deCategoriaAnterior: string;

    { Variáveis configuradas com os valores da linha
      do banco que está sendo manipulada }
    deCategoriaAtual: string;
    deProdutoAtual: string;
    valorAtual: Double;
    unidadesAtual: integer;

    { Variáveis Totalizadoras da Categoria Atual }
    valorCategoriaAtual: Double;
    valorEstoqueCategoriaAtual: Double;
    quantidadeCategoriaAtual: integer;

    { Variáveis Totalizadoras do "Total Geral" }
    valorTotal: Double;
    valorEstoqueTotal: Double;
    quantidadeTotal: integer;

    { Variável para controlar a linha atual,
      deve ser inicializada com 1 }
    indiceLinha: integer;

    { DataTable que conterá os dados manipulados e
      servirá de fonte de dados para o DataGrid }
    dtDadosAgrupados: DataTable;
```

Listagem 6. Método CriaDataTableDadosAgrupados

```
{ Cria o DataTable que conterá os dados manipulados
  em forma de subgrupos }
procedure TWebForm1.criaDataTableDadosAgrupados;
begin
  dtDadosAgrupados := DataTable.Create;
 dtDadosAgrupados.Columns.Add('Descricao',
    System.Type.GetType('System.String'));
 dtDadosAgrupados.Columns.Add('valorUnitario',
    System.Type.GetType('System.Double'));
 dtDadosAgrupados.Columns.Add('unidadesEstoque',
    System.Type.GetType('System.Int32'));
 dtDadosAgrupados.Columns.Add('valorEstoque',
    System.Type.GetType('System.Double'));
  dtDadosAgrupados.Columns.Add('tipoLinha',
    System.Type.GetType('System.Int32'));
end;
```

Listagem 7. Método populaDataTableDadosAgrupados

```
{ Navega no DataTable retornado no SQL e popula o
  DataTable Agrupado (no formato final) }
procedure TWebForm1.populaDataTableDadosAgrupados;
var
  drRetornoBD: DataRow;
begin
  for drRetornoBD in
    retornaCategoriasProdutos().Rows do
  begin
    { salva os dados da linha atual para serem
      manipulados nos outros métodos }
    deCategoriaAtual := drRetornoBD['Categoria'].ToString;
    deProdutoAtual := drRetornoBD['Produto'].ToString;
    valorAtual := Convert.ToDouble(drRetornoBD['valorUnitario']);
    unidadesAtual := Convert.ToInt32(drRetornoBD['unidadesEstoque']);

    { Valida troca de categoria }
    if (deCategoriaAnterior <> deCategoriaAtual) then
    begin

      { Insere subTotal da Categoria anterior, porém
      somente se índiceLinha <> 1, pois nesse caso
      seria a primeira iteração no for. }
       if (indiceLinha <> 1) then
         dtDadosAgrupados.Rows.Add(
        retornaLinhaSubTotal);

       { adiciona linha com a descrição da nova Categoria }
      dtDadosAgrupados.Rows.Add(retornaLinhaCategoria);

      { "true" para zerar os somatórios dessa
        categoria, pois mudou a categoria. }
      atualizaTotalizadores(True);

      { atualiza deCategoriaAnterior com a Atual }
      deCategoriaAnterior := deCategoriaAtual;
    end;

   { Adiciona linha de Produto no DataTable agrupado }
   dtDadosAgrupados.Rows.Add(retornaLinhaProduto);

   { Incrementa Totalizadores e "false" para não zerar
     os somatórios desta categoria }
   atualizaTotalizadores(False);
   Inc(indiceLinha);
  end;

   { adiciona subtotal da última categoria }
  dtDadosAgrupados.Rows.Add(retornaLinhaSubTotal);

  { adiciona Total Geral }
   dtDadosAgrupados.Rows.Add(retornaLinhaTotal);
end;
```

(**Listagem 6**). Atente para as colunas *Descrição* e *tipoLinha*. Para permitir que seja feito o agrupamento, a coluna *Descricao* possuirá as descrições das Categorias, Produtos, subtotais e Total Geral, e por sua vez a coluna *tipoLinha* informará ao *DataTable* o tipo da linha conforme a enumeração *TipoLinha*, declarado anteriormente.

Através do *retornaCategoriasProdutos*, obtêm-se o *DataTable* com o retorno do banco, possibilitando o preenchimento do *dtDadosAgrupados* já no formato a ser mostrado ao usuário, isso é, com grupos de Categorias, subtotais e Total Geral.

Digite o *populaDataTableDadosAgrupados* (**Listagem 7**), que então varre o retorno do banco e faz a validação de troca de categorias.

Caso o registro atual da iteração seja uma nova categoria, são adicionadas as linhas de *Subtotal* (da categoria anterior) e a linha de *Descrição* da nova categoria. No final, é adicionada a linha de *Total Geral*.

Observe que em cada iteração, são atualizados os totalizadores dos subtotais e Total Geral, sendo que na troca de categoria, os subtotais são zerados. A seguir, devem ser adicionados os métodos que retornam as linhas descritivas das Categorias, Produtos, subtotais e Total Geral ao *dtDadosAgrupados* (**Listagem 8**).

Listagem 8. Métodos para retornar as linhas descritivas

```
{ Retorna o DataRow com a Descrição da Categoria Atual }
function TWebForm1.retornaLinhaCategoria: DataRow;
var
  drCategoria: DataRow;
begin
  { Linha de subtitulo da categoria }
  drCategoria := dtDadosAgrupados.NewRow;
  drCategoria := dtDadosAgrupados.NewRow;
  drCategoria['Descricao'] := ' >>Categoria: '+ deCategoriaAtual;
  drCategoria['tipoLinha'] := TipoLinha.DescricaoCategoria;
  { fim linha de subtitulo da categoria }
  Result := drCategoria;
end;

{ Retorna o DataRow com Dados do Produto Atual }
function TWebForm1.retornaLinhaProduto: DataRow;
var
  drProduto: DataRow;
begin
  { Retorna a linha com os dados do produto }
  drProduto := dtDadosAgrupados.NewRow;
  drProduto['Descricao'] := deProdutoAtual;
  drProduto['valorUnitario'] := &Object(valorAtual);
  drProduto['unidadesEstoque'] := &Object(unidadesAtual);
  drProduto['valorEstoque'] := &Object(valorAtual * unidadesAtual;
  drProduto['tipoLinha'] := TipoLinha.DescricaoProduto;
  Result := drProduto;
end;

{ Retorna DataRow com SubTotal da Categoria }
function TWebForm1.retornaLinhaSubTotal: DataRow;
var
  drSubTotal: DataRow;
begin
  { Retorna a linha de subtotal após cada categoria }
  drSubTotal := dtDadosAgrupados.NewRow;
  drSubTotal := dtDadosAgrupados.NewRow;
  drSubTotal['Descricao'] := 'Sub Total';
  drSubTotal['valorUnitario'] := &Object(
    valorCategoriaAtual);
  drSubTotal['unidadesEstoque'] := &Object(quantidadeCategoriaAtual);
  drSubTotal['valorEstoque'] := &Object(valorEstoqueCategoriaAtual);
  drSubTotal['tipoLinha'] := TipoLinha.SubTotal;
  Result := drSubTotal;
end;

{ Retorna DataRow com Total Geral }
function TWebForm1.retornaLinhaTotal: DataRow;
var
  drTotal: DataRow;
begin
  { Retorna a linha final de Total Geral }
  drTotal := dtDadosAgrupados.NewRow;
  drTotal['Descricao'] := 'Total Geral';
  drTotal['valorUnitario'] := &Object(valorTotal);
  drTotal['unidadesEstoque'] := &Object(quantidadeTotal);
  drTotal['valorEstoque'] := &Object(valorEstoqueTotal);
  drTotal['tipoLinha'] := TipoLinha.Total;
  Result := drTotal;
end;
```

Listagem 9. Método atualizaTotalizadores

```
procedure TWebForm1.atualizaTotalizadores(
  _zeraTotCampanhaAtual: Boolean);
begin
  if _zeraTotCampanhaAtual then
  begin
    { Zera totalizadores da Categoria Atual }
    valorCategoriaAtual := 0;
    quantidadeCategoriaAtual := 0;
    valorEstoqueCategoriaAtual := 0;
  end
  else
  begin
    { Incrementa totalizadores da Categoria Atual }
    valorCategoriaAtual := valorCategoriaAtual + valorAtual;
    quantidadeCategoriaAtual :=
      quantidadeCategoriaAtual + unidadesAtual;
    valorEstoqueCategoriaAtual :=
      valorEstoqueCategoriaAtual +
      (valorAtual * unidadesAtual);
    { Incrementa totalizadores do Total Geral }
    valorTotal := valorTotal + valorAtual;
    quantidadeTotal := quantidadeTotal + unidadesAtual;
    valorEstoqueTotal := valorEstoqueTotal +
      (valorAtual * unidadesAtual);
  end;
end;
```

Listagem 10. Page_Load

```
if (not IsPostBack) then
begin
  { Deve ser inicializada com 1 }
  indiceLinha := 1;
  { cria DataTable com os DadosAgrupados }
  criaDataTableDadosAgrupados;
    { popula o DataTable Agrupado, através dos dados
    provenientes da consulta SQL no Banco de Dados }
    populaDataTableDadosAgrupados;
    { configura a fonte de dados e mostra os dados }
    gridProdutoCategorias.DataSource :=
    dtDadosAgrupados;
   gridProdutoCategorias.DataBind();
 end;
 { Remove o CheckBox entre postbacks }
 removeCheckBox;
```

Listagem 11. Método removeCheckBox

```
{ Remove o CheckBox das linhas Totalizadoras }
procedure TWebForm1.removeCheckBox;
var
  item: DataGridItem;
begin
  { Remove o CheckBox nas linhas de categoria,
    subtotais e Total Geral }
  for item in gridProdutoCategorias.Items do
  begin
    if ((item.Cells[1].Text = 'Sub Total') or
      (item.Cells[1].Text = 'Total Geral') or
      (item.Cells[1].Text.ToString().IndexOf(
      '>>Categoria:') <> -1)) then
        item.Cells[0].Controls.RemoveAt(0);
  end;
end;
```

Listagem 12. ItemDataBound do DataGrid

```
var
  drvItem: DataRowView;
  tipo: integer;
begin
  if (e.Item.ItemType = ListItemType.Item) or
    (e.Item.ItemType =
    ListItemType.AlternatingItem) then
 begin
    { drvItem faz refêrencia a linha do DataSource
     do DataGrid do item atual }
    drvItem := (e.Item.DataItem as DataRowView);
    tipo := Convert.ToInt32(drvItem['tipoLinha']);
    { colorir de acordo com o tipo de linha }
    case tipo of
      { Categoria }
      0: e.Item.BackColor := Color.Silver;
      { SubTotal }
      2: e.Item.BackColor := Color.Gainsboro;
      { Total }
      3: e.Item.BackColor := Color.Gray;
    end;
  end;
end;
```

Para finalizar a implementação do preenchimento do *DataTable* agrupado, é necessário adicionar o *atualizaTotalizadores* (**Listagem 9**), que incrementa ou zera os totalizadores da categoria da iteração atual e que também incrementa os totalizadores do Total Geral.

Preenchido o *DataTable* agrupado, altere o *Page_Load* (**Listagem 10**), para que efetivamente faça a chamada aos métodos que criam e preenchem o *DataTable*, além de fornecê-lo ao *DataGrid* como seu *DataSource*.

Todo código no "*not IsPostBack*" será executado apenas a primeira vez em que página for carregada.

Para cada item que for preenchido ao *DataGrid*, será criado um *CheckBox*, devido à coluna *TemplateColumn*. Porém, para as linhas de Categorias, subtotais e Total Geral, não é desejado mostrá-lo ao usuário, justificando o uso de *removeCheckBox* (**Listagem 11**) referenciada no *Page_Load*.

No momento que é realizado o *DataBind* do *DataGrid*, automaticamente é disparado o evento *ItemDataBound*, que ocorre como última oportunidade para acessar o dado de um item do *DataGrid*, antes que seja exposto ao usuário.

Dessa forma, o utilizaremos para colorir as linhas totalizadoras. Para criar esse evento, selecione o *gridProdutoCategorias*, na janela de propriedades dê um duplo clique no *ItemDataBound* e digite o código da **Listagem 12**.

Note que verificado o item atual do *DataGrid*, encontra-se, através do *DataRowView*, o valor da coluna *tipoLinha* do *dtDadosAgrupados*. Através desse valor e do enumerador *TipoLinha*, colore-se o item do *DataGrid*.

Com o objetivo de mostrar que é possível capturar dados do *DataGrid* entre *postbacks*, mesmo que suas colunas tenham sido criadas dinamicamente, dê um duplo clique no *btnSelecionados* e digite o código da **Listagem 13**, que listará no *ListBox* todos os itens que o *CheckBox* estiver selecionado.

Salve o projeto e execute para ver o resultado do *gridProdutoCategorias* montado dinamicamente. Experimente selecionar alguns *CheckBoxes*, clique no *Mostrar Selecionados* e veja como os itens permanecem selecionados e também listados no *ListBox*. A aplicação final em execução é mostrada na **Figura 3.**

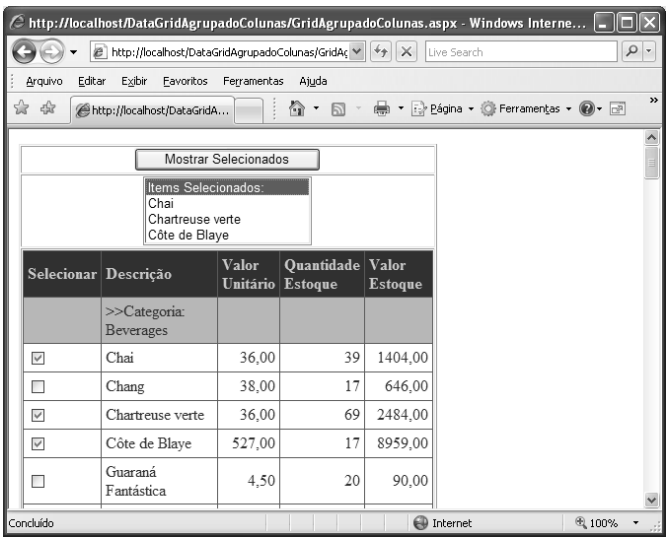


Figura 3. Grid com colunas criadas dinamicamente

Conclusões

A combinação entre *DataGrid* e *DataTable* permitem ao desenvolvedor manipular dados e mostrá-los aos usuários de uma forma muito dinâmica e eficiente. A criação de colunas dinamicamente viabilizam a solução de problemas quanto a formatação do *DataGrid*, aumentando em muito sua aplicabilidade, como por exemplo, um sistema de geração de relatórios. ■

Fale conosco!

Conheça a Central de Atendimento on-line da DevMedia. Agora ficou muito mais fácil de acompanhar sua assinatura, alterar seus dados cadastrais e retirar qualquer dúvida sobre nossos serviços e produtos. Para isso basta ter em mãos seu login e senha.

Na Central de Atendimento você:

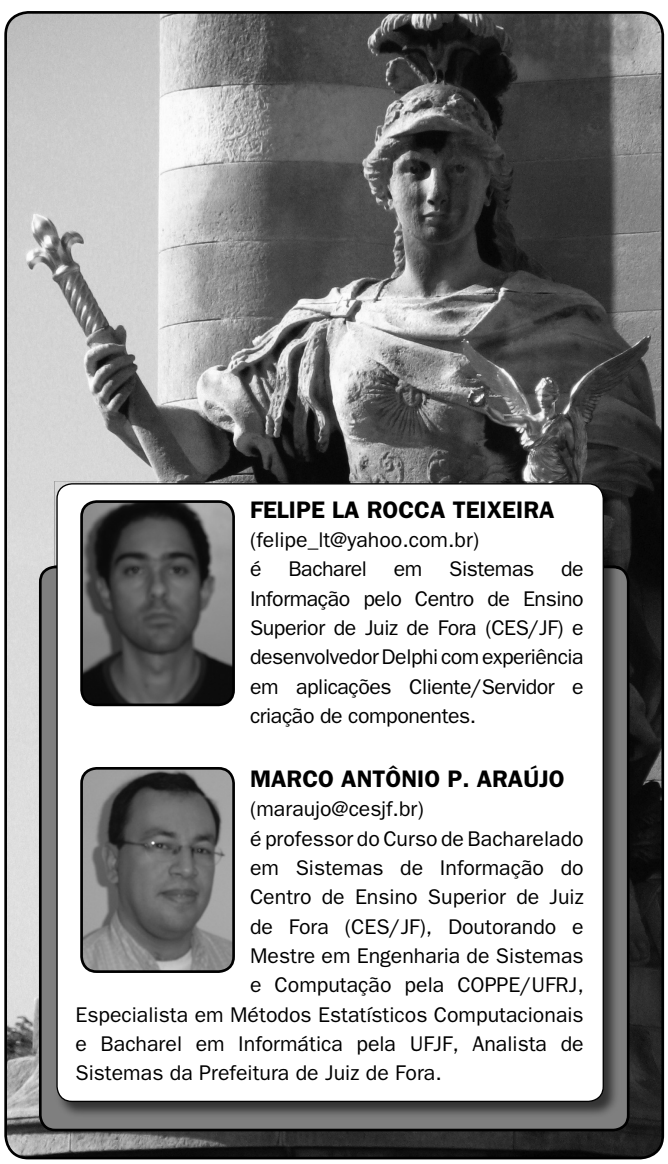
- Consulta as perguntas mais frequentes
- Renova por boleto ou cartão de crédito
- Consulta e altera seus dados cadastrais
- Consulta a data de envio de cada exemplar

DevMedia GROUP

Para entrar em contato com a DevMedia, basta postar sua dúvida na própria Central de Atendimento através de um sistema web, prático, seguro e fácil de usar. Com essa nova ferramenta você tem a garantia de receber suas respostas em menor tempo e com alto padrão de qualidade.

www.devmedia.com.br/central

Análise de Requisitos com CaliberRM



FELIPE LA ROCCA TEIXEIRA
(felipe_lt@yahoo.com.br)
é Bacharel em Sistemas de Informação pelo Centro de Ensino Superior de Juiz de Fora (CES/JF) e desenvolvedor Delphi com experiência em aplicações Cliente/Servidor e criação de componentes.

MARCO ANTÔNIO P. ARAÚJO
(maraujo@cesjf.br)
é professor do Curso de Bacharelado em Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora (CES/JF), Doutorando e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ, Especialista em Métodos Estatísticos Computacionais e Bacharel em Informática pela UFJF, Analista de Sistemas da Prefeitura de Juiz de Fora.

Visando prover os subsídios necessários para produzir software de alta qualidade a um baixo custo e aumentar a produtividade no processo de desenvolvimento, a Engenharia de Software pode ser vista como o estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja mais confiável e que funcione eficientemente.

Diversos são os desafios encontrados pelos desenvolvedores ao longo do ciclo de vida de um produto de software. Métodos, processos, técnicas e ferramentas devem ser integrados no intuito de apoiar o desenvolvimento de sistemas. A Análise de Requisitos é uma das primeiras atividades técnicas nesse processo, e responsável por definir as funcionalidades de um sistema e sob que restrições o sistema deve funcionar, levando em consideração o ponto de vista de satisfação das necessidades do usuário final.

Mesmo assim, estudos demonstram que grande parte de projetos de software não terminam ou fracassam por não atenderem completamente às necessidades dos usuários, ou por excederem os orçamentos previstos e os prazos estimados.

Por isso, fazer um bom levantamento e especificação de requisitos é muito importante, visto que problemas de análise requisitos surgem por diversos motivos:

- Em geral *stakeholders* (os participantes do projeto), não sabem exatamente o que querem ou não expressam bem os requisitos do sistema;
- Atender as necessidades de diferentes *stakeholders* pode gerar requisitos conflitantes;
- Requisitos mudam durante o processo de desenvolvimento, *stakeholders* novos podem surgir e o ambiente de trabalho mudar;
- Fatores políticos e organizacionais podem influenciar os requisitos do sistema, uma vez que, quem tem o poder de decisão pode impor diferentes requisitos ao sistema.

Então, torna-se difícil manter sob controle o conjunto de requisitos de um sistema, mesmo diante de alterações inevitáveis. Na prática, a gerência de requisitos é fundamental para que essas mudanças sejam controladas e, através de um planejamento, devem-se adotar políticas para a identificação de requisitos e gerenciamento de mudanças.

A ferramenta CaliberRM

CaliberRM é uma ferramenta de automação de gerenciamento de requisitos, que funciona em ambiente cliente/servidor. Utilizado no desenvolvimento de projetos de software, visa facilitar a colaboração, análise de impacto devido às mudanças e o gerenciamento de alterações de requisitos, ajudando os participantes do projeto a aumentar a qualidade das aplicações, sua produtividade e a satisfação dos usuários finais.

Essa ferramenta não trata os requisitos como simples documentos, mas sim como um grupo de objetos integrados e reutilizáveis, além de contar com um repositório central, permitindo a comunicação entre todos os envolvidos no projeto.

Permite também a integração com outras ferramentas, como o Delphi, facilitando a relação entre os requisitos e o código-fonte da aplicação.

Nota: *se estiver utilizando a versão CaliberRM Release 2 SP1, para que funcione a integração com o Delphi 2006 é necessário efetuar o download da atualização* Hotfix 2 no endereço *www.borland.com/downloads/registered/download_delphi.html*.

Criando um exemplo

Para a aplicação de exemplo, deseja-se desenvolver um sistema para uma biblioteca. No intuito de demonstrar os passos para especificação de requisitos utilizando o CaliberRM, serão abordados somente alguns requisitos funcionais desse sistema.

Requisitos Funcionais descrevem funcionalidades e serviços do sistema, definindo como esse deve reagir e se comportar em diversas situações, enquanto Requisitos Não Funcionais descrevem restrições sobre funções ou serviços oferecidos pelo sistema, como desempenho, resposta ao usuário, segurança, precisão, confiabilidade e robustez, dentre outros.

Primeiramente deve-se criar um novo projeto utilizando a ferramenta de configuração da CaliberRM, o Framework Administrator. Ao abrir o programa será necessário efetuar o login como Administrador. Para isso, deve-se informar para o campo *Server* o nome do computador e para os campos *User* e *Password* a palavra "*admin*".

Esse usuário é criado automaticamente na instalação da CaliberRM e a partir dele é possível criar novos grupos e usuários de acesso. A seguir, deve-se criar o novo projeto com o nome de "Biblioteca" através do menu *File>New Project*, e definir a integração com outras ferramentas na aba *External Traceability*, selecionando para a lista *Enabled*, as opções *BorlandCaliberAddin* e *File* (**Figura 1**).

Nota: *para a utilização da ferramenta Framework Administrator é necessário a instalação da* **CaliberRM Server**. *O download pode ser feito em* www.borland.com/downloads/download_caliber.html.

Definindo os requisitos do sistema

Com o projeto criado, já podemos começar a especificar os requisitos do sistema proposto utilizando o CaliberRM através do *Iniciar>Programas>CaliberRM>CaliberRM*. Ao abrir o programa selecione o projeto *Biblioteca* que, em seguida, abrirá automaticamente a janela de Tipos de Requisitos.

Ou então clique com o botão direito selecionando *Set Requirement Types*. Por *default*, o CaliberRM exibe uma lista pré-definida de tipos de requisitos, mas podem-se criar novos tipos ou simplesmente modificá-los. Para este exemplo, marque *Functional Requirements* para especificar os requisitos funcionais (**Figura 2**).

O próximo passo será criar os requisitos do sistema, isso tanto pode ser feito utilizando a própria ferramenta CaliberRM ou pelo Delphi 2006. O Delphi possui uma interface *client* de integração com o CaliberRM permitindo visualizar, criar e modificar os requisitos de um projeto.

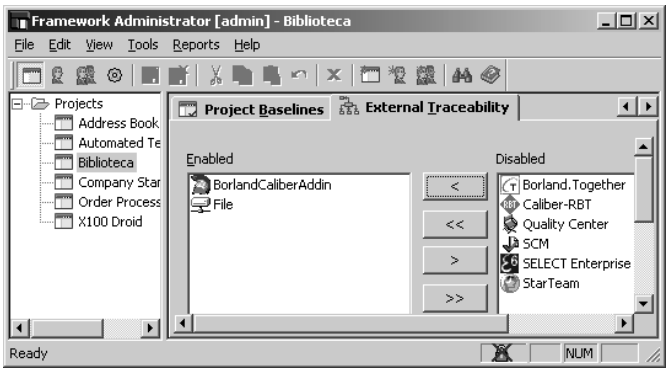


Figura 1. Visualização do Framework Administrator

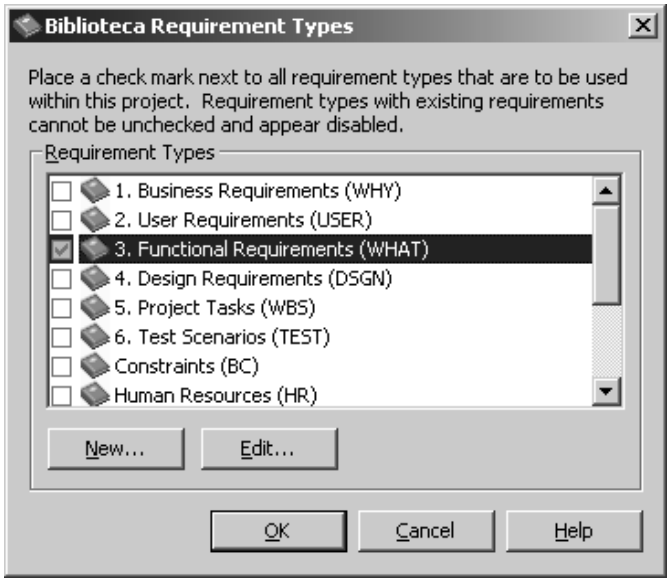


Figura 2. Visualização da janela de Tipos de Requisitos

Utilizando o Delphi 2006, deve-se acessar o menu *View>CaliberRM Requirements* e selecionar o projeto *Biblioteca* e o tipo de requisito *Functional Requirements*, anteriormente incluído, e clicar com o botão direito selecionando *Create Requirement>Child*.

Depois, na aba *Details*, deve-se preencher o nome do requisito como "Cadastrar Usuário" e sua descrição, selecionando então o valor *Submitted* para a opção *Status*, *Essential* para *Priority* e *admin* para *Owner* (**Figura 3**). Para criar o requisito *Efetuar Empréstimo*, deve-se proceder de maneira semelhante.

Deve-se observar que existem várias opções para a definição de um requisito através das diferentes abas:

- *Details*: define o nome do requisito, seu responsável, estado, prioridade e a versão que é alterada automaticamente sempre que ocorrer alguma modificação;
- *Traceability*: permite o rastreamento de requisitos e artefatos, identificando possíveis incoerências devido às alterações. O CaliberRM cria *links* entre requisitos e artefatos, assim pode-se ter rastreabilidade de/para o requisito selecionado;
- *Validation*: permite definir um procedimento de validação para cada requisito através de um campo de texto;
- *Discussion*: utilizado para troca de mensagens entre os participantes do projeto, ficando todas as mensagens registradas para futuras consultas;
- *References*: permite adicionar referências aos requisitos, podendo ser arquivos externos, de texto ou um link para uma URL;
- *History*: exibe uma lista de todas as alterações que foram feitas no requisito e a descrição de cada alteração. Assim é possível consultar toda a evolução do requisito, obtendo dados como datas, responsável pela alteração, comentários e o que foi alterado;
- *Responsibilities*: permite visualizar e alterar todos os grupos e usuários que estão envolvidos com o requisito;
- *User Attributes*: estão os atributos referentes a cada tipo de requisito;
- *Ambiguity Review*: informa ambigüidades encontradas por diferentes usuários no requisito;
- *Estimate*: permite incluir informações para que se possa futuramente gerar um gráfico com estimativas de esforço.

Implementando requisitos

A integração do CaliberRM com o IDE do Delphi, contribui bastante para o processo de desenvolvimento, facilitando o entendimento das modificações, evitando o re-trabalho e aumentando a produtividade da equipe de projeto.

Levando em consideração que os outros requisitos do sistema de *Biblioteca* já tenham sido implementados, como os cadastros de publicações e de exemplares, dentre outros, a implementação dos novos requisitos para o cadastro de usuários e efetuar empréstimo podem ser visualizadas nas **Figuras 4** e **5**, respectivamente.

Para que possamos realizar uma análise de impacto de futuras alterações, por exemplo, no requisito *Cadastrar Usuário*, deve-se selecionar esse requisito e, na aba *Traceability*, lista *Traces To*, incluir os arquivos *CadastrarUsuario.pas* e *CadastrarUsuario.dfm*, bem como o requisito *Efetuar Empréstimo* (**Figura 6**).

Dessa forma tem-se que qualquer modificação realizada no requisito *Cadastrar Usuário* poderá implicar em modificações nesses artefatos.

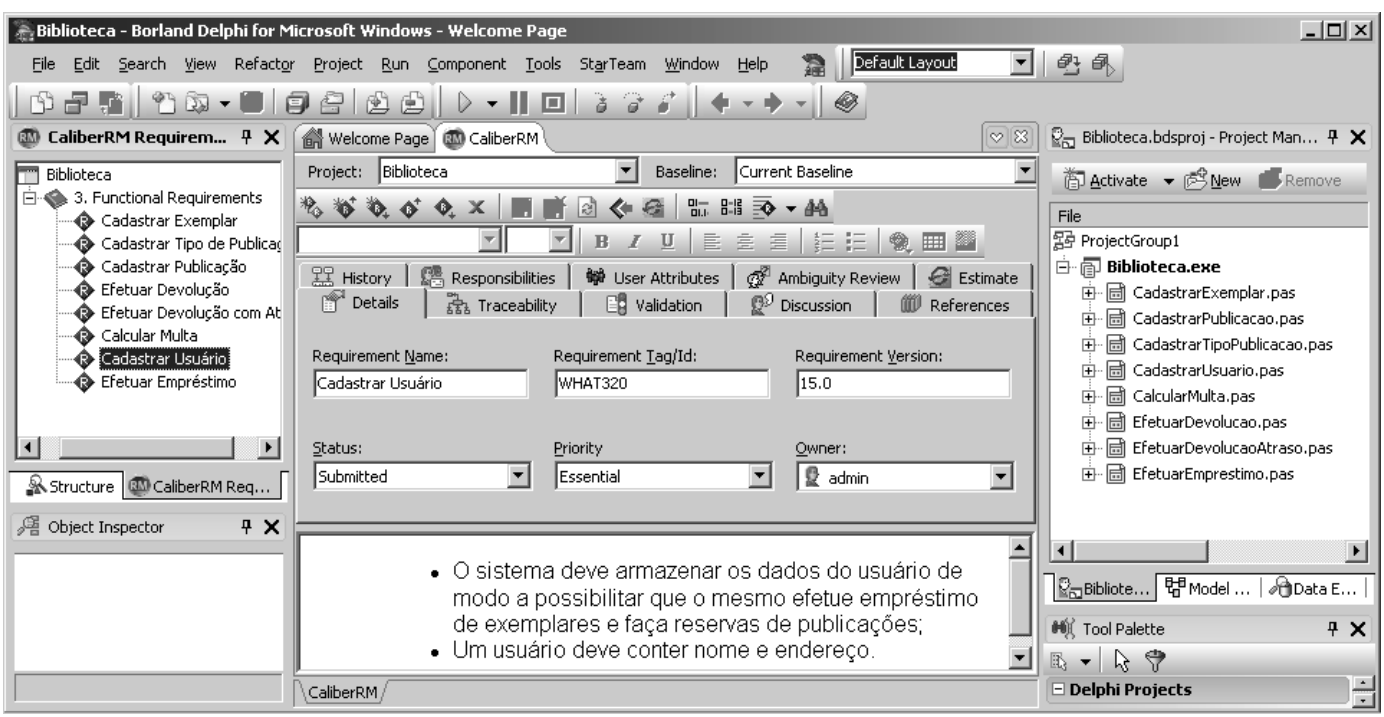


Figura 3. Requisito Cadastrar Usuário no CaliberRM Requirements

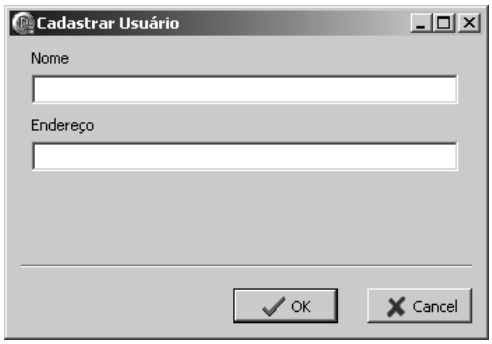


Figura 4. Tela para cadastrar usuário

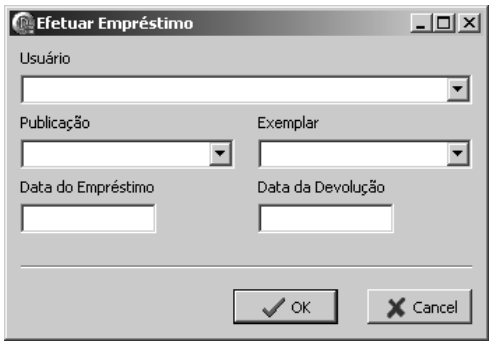


Figura 5. Tela para efetuar empréstimo

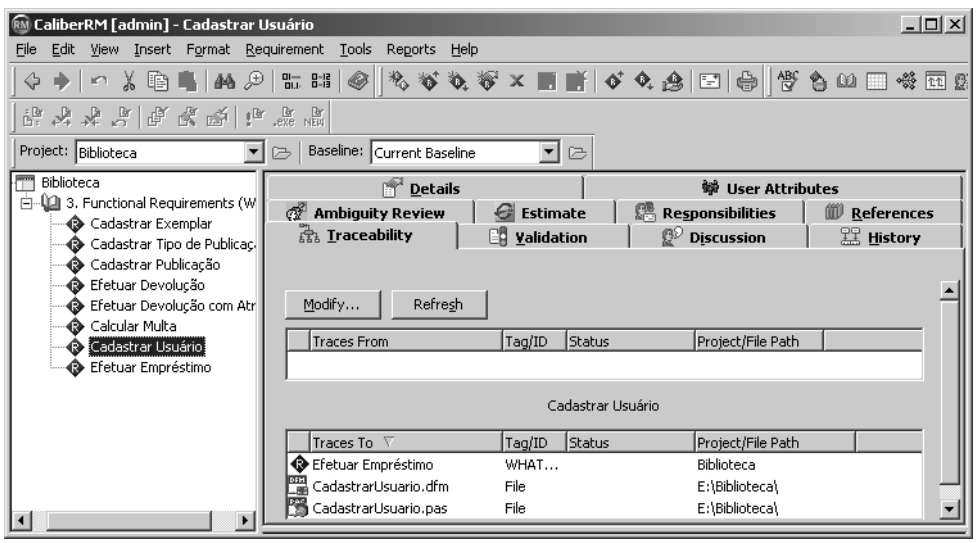


Figura 6. Vizualização da aba *Traceability* do requisito Cadastrar Usuário

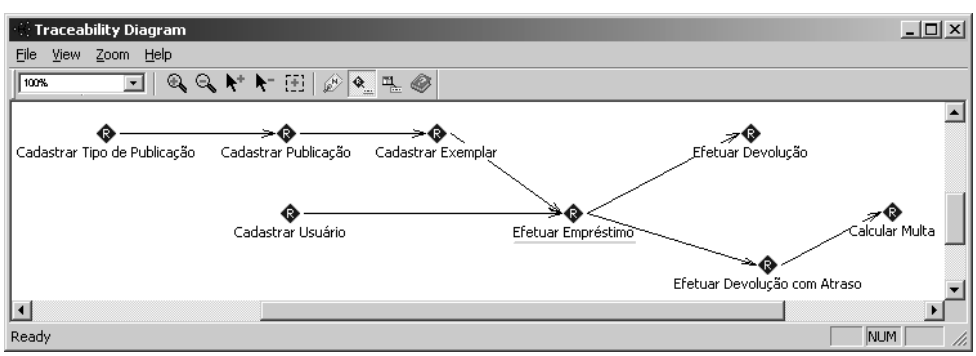


Figura 7. Traceability Diagram dos requisitos a partir do requisito Efetuar Empréstimo

Assim, deve-se fazer o mesmo para o requisito *Efetuar Empréstimo*, incluindo os arquivos *EfetuarEmprestimo.pas* e *EfetuarEmprestimo.dfm* na sua lista *Traces To*, aparecendo automaticamente na lista *Traces From* o requisito *Cadastrar Usuário*, demonstrando o relacionamento entre os dois requisitos.

Em *Tools>Traceability Diagram* pode-se visualizar a rastreabilidade dos requisitos do sistema a partir do requisito *Efetuar Empréstimo* (**Figura 7**).

Na aba *History* do requisito *Cadastrar Usuário*, por exemplo, pode-se verificar toda a evolução de modificações sofridas pelo requisito. A ferramenta CaliberRM salva todas as versões geradas, facilitando assim futuras consultas e comparações, contribuindo em muito para o controle da evolução do requisito.

De forma prática é possível identificar todo o histórico de modificações sofridas pelo requisito selecionando a versão desejada (**Figura 8**).

Gerência de Requisitos

Através do *Traceability Diagram*, pode-se colocar em prática conceitos de Gerência de Requisitos, acompanhando a evolução dos requisitos ao longo do processo de desenvolvimento, fazendo com que participantes do projeto possam compreender rapidamente o impacto de alterações e conseqüências geradas por modificações.

Em nosso exemplo, se for necessária a inclusão de novos campos no cadastro de usuários, como telefones, e-mail e se o usuário está ativo ou não, deve-se alterar o requisito *Cadastrar Usuário* incluindo as novas características solicitadas no campo descrição da aba *Details*.

Pode-se verificar que essa modificação afetará diretamente ou indiretamente outros requisitos, bem como outros artefatos (**Figura 9**), como por exemplo arquivos fonte (.PAS).

O *Traceability Diagram* identifica através de setas em vermelho com um ponto de interrogação que os artefatos *CadastrarUsuario.pas* e *CadastrarUsuario.dfm*, e o requisito *Efetuar Empréstimo*, poderão sofrer alterações, pois poderão ter que considerar essas novas características do requisito *Cadastrar Usuário*.

Na aba *Traceability* do requisito *Cadastrar Usuário* também é possível identificar essa modificação que acabou de ocorrer no requisito, na lista *Traces To* os artefatos listados receberão para o campo *Status* a palavra *Suspect*, informando que esses poderão sofrer alguma alteração.

Dessa forma, é possível fazer uma análise de impacto mais precisa, fazendo com que a equipe de projeto e até o próprio cliente possam visualizar os efeitos dessa nova solicitação, apresentando uma estimativa mais precisa de esforço, prazos e custos necessários para a mudança.

Outras funcionalidades do CaliberRM

O CaliberRM ainda oferece outras soluções como rastreabilidade entre todos os artefatos, estimativa profissional, criação de *baseline*, geração de documentos através do *Document Factory*, efetuar comparações entre diferentes requisitos e en-

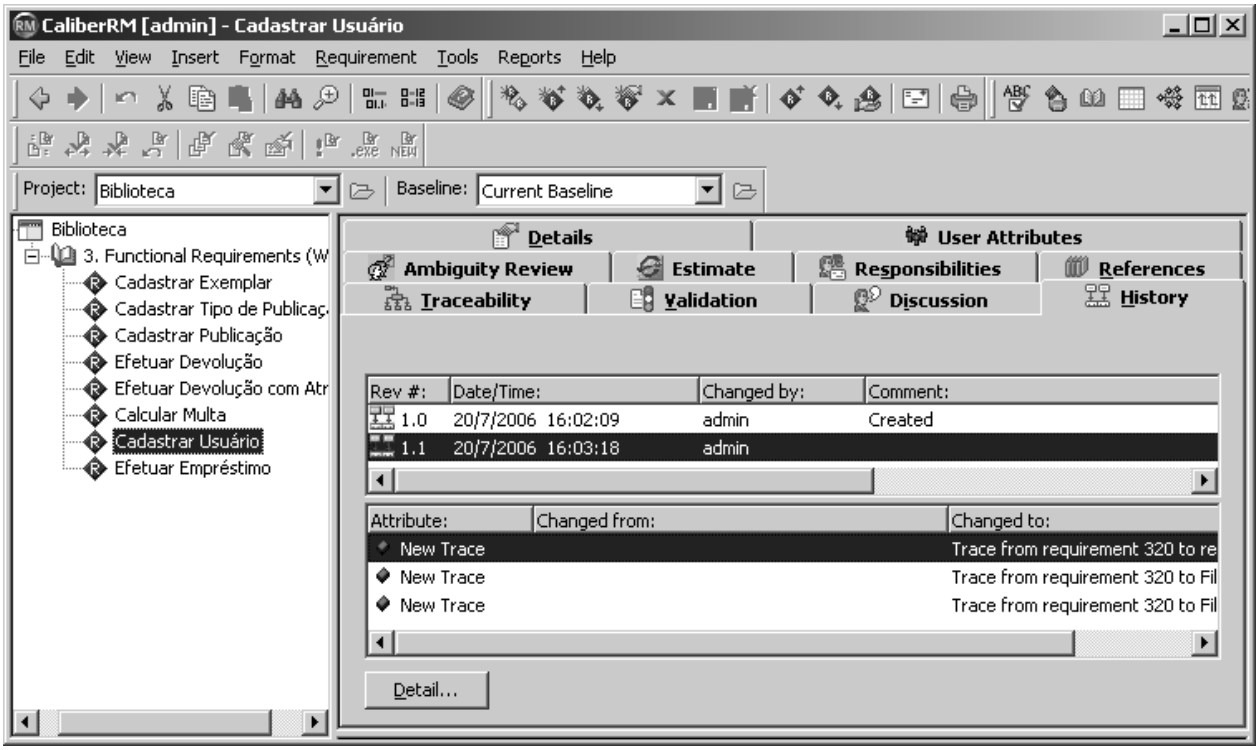


Figura 8. History do requisito Cadastrar Usuário

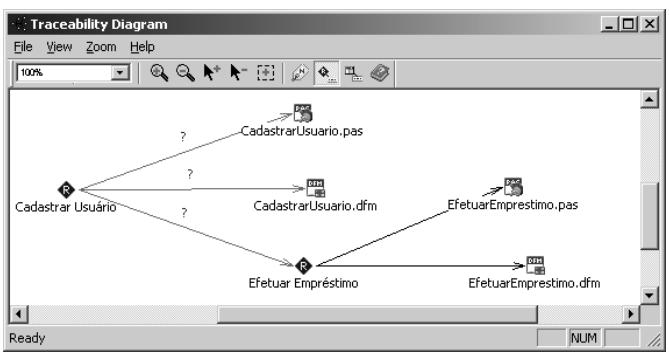


Figura 9. Traceability Diagram do requisito Cadastrar Usuário

Figura 10. Project Baseline no Framework Administrator

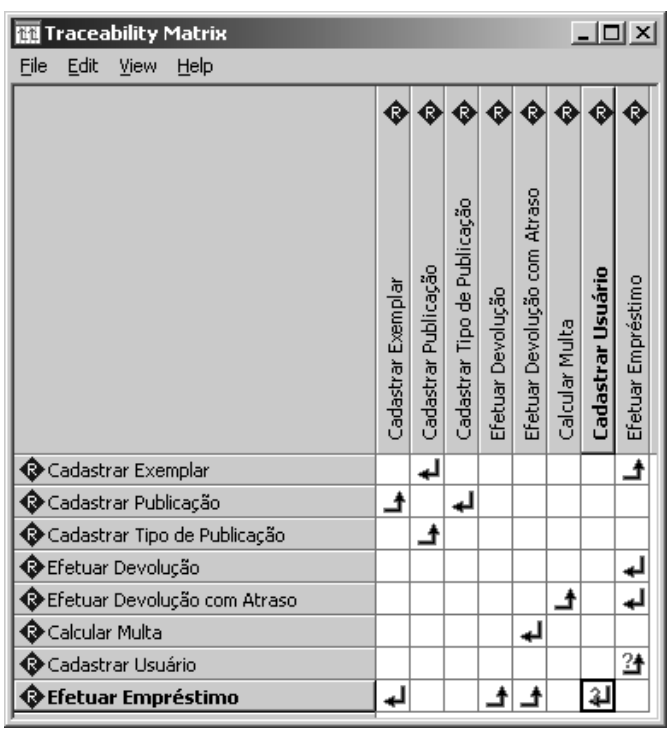


Figura 11. Visualização do Traceability Matrix

tre versões de um mesmo requisito, e vários relatórios de detalhes dos requisitos, histórico, responsabilidades, discussões e status, proporcionando um controle efetivo dos requisitos durante o andamento do projeto.

Com o *Document Factory*, disponível em *Tools>Document Factory*, é possível gerar documentos de requisitos em um formato especificado. Para isso, pode-se definir um *template* no Word para formatar a saída das informações dos requisitos de acordo com as necessidades desejadas.

Uma **baseline** é um conjunto de artefatos aceitos e controlados, e que serão utilizados em atividades posteriores à sua criação, para servir de mecanismo de comparação entre versões de projetos. A criação de uma *baseline* pode ser feita quando ocorrer uma grande mudança na estrutura dos requisitos do projeto, como a criação de um novo conjunto requisitos.

Para criar uma nova *baseline* deve-se utilizar o Framework Administrator selecionar a aba *Project Baseline* do projeto *Biblioteca*, clicar em *New* e preencher a tela com as informações da nova *baseline* (**Figura 10**).

Em *Tools>Traceability Matrix*, tem-se uma visualização completa da rastreabilidade definida entre todos os artefatos do projeto, ajudando na verificação e validação dos requisitos. Através dele, também se pode verificar quais artefatos foram afetados com a nova solicitação, através de um ponto de interrogação em vermelho (**Figura 11**).

O relatório *Detail*, disponível em *Reports>Detail*, é muito interessante, pois através dele é possível ter uma visão completa de todas as características dos requisitos do projeto, como descrição, responsável, estado, prioridade, versão e seus atributos, além de possuir opção para impressão do histórico de modificações e discussões geradas sobre os requisitos (**Figura 12**).

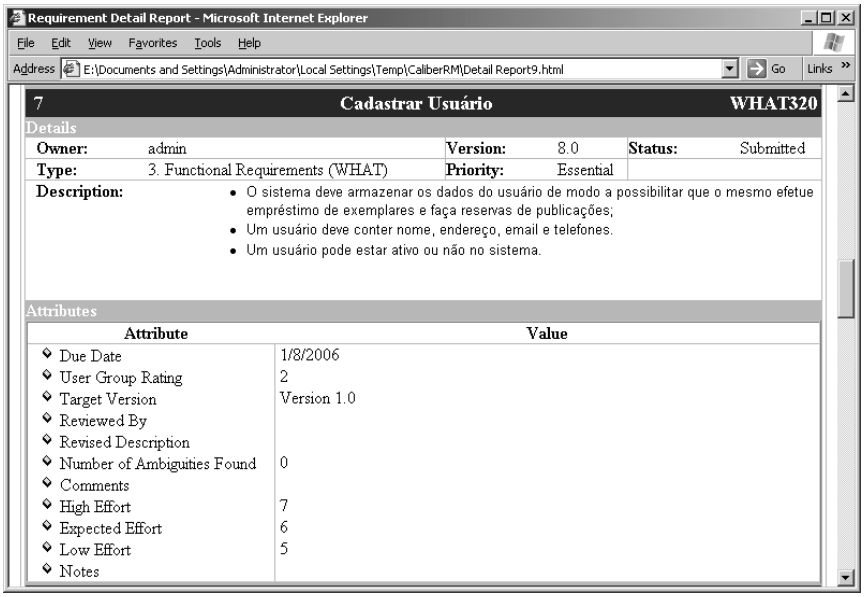
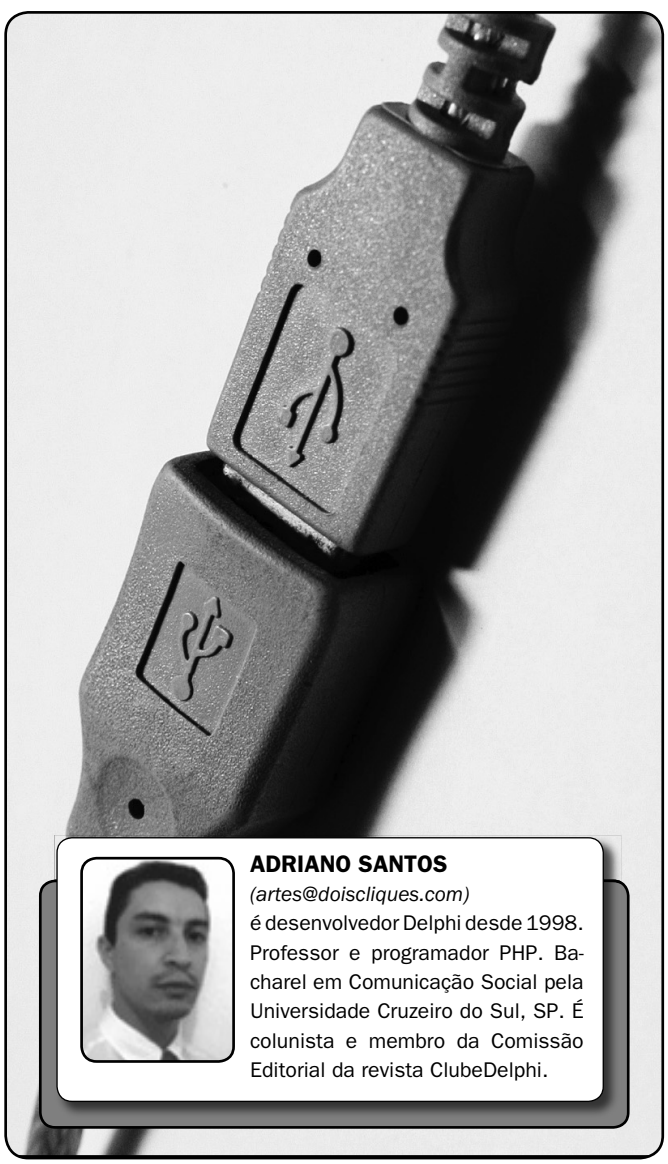


Figura 12. Visualização do relatório Detail

Conclusões

Vimos neste artigo como pode ser útil o uso de uma ferramenta de gerenciamento de requisitos durante o desenvolvimento de projetos de software. O CaliberRM demonstra ser uma ferramenta poderosa e pronta para atender às necessidades do processo da Análise de Requisitos, permitindo um efetivo processo de gerência de requisitos e auxiliando na garantia da qualidade. ■

Troca eletrônica de dados por EDI



ADRIANO SANTOS
(artes@doiscliques.com)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É colunista e membro da Comissão Editorial da revista ClubeDelphi.

EDI (*Eletronic Data Interchange*) ou Troca Eletrônica de Dados; a sigla pode não ser tão conhecida, mas certamente sua utilização e conceito proporcionam aos desenvolvedores a oportunidade de integrar informações de maneira rápida e confiável.

Uma das principais vantagens em se desenvolver EDIs está em poder efetuar a troca de dados entre organizações distintas, ou seja, podemos facilmente enviar dados relevantes entre cliente e fornecedor de maneira clara e estruturada.

Outra grande vantagem aplica-se ao fato de que é possível integrar grandes volumes de informações com os mais variados bancos de dados, desde os antigos Paradox e DBase até Oracle, Firebird ou mesmo PostgreSQL.

Por exemplo: uma empresa operando com Firebird 1.5 pode, enviar registros de notas ficais para sua empresa de contabilidade que opera com um Paradox ou solicitar à instituição financeira onde possui conta corrente, que gere os boletos bancários do mês atual.

Mas o que são exatamente os EDIs?

Arquivos texto em formato e padrão pré-estabelecido normalmente pela instituição que o receberá e, consequentemente, o carregará. Há basicamente duas formas de compor esses arquivos: usando tabulações ou delimitadores, tais como ponto-e-vírgula ou pipe-line.

O sistema bancário é um dos grandes utilizadores desse mecanismo. Chamado de *cobrança escritural*, os arquivos texto trocados entre empresas e bancos contém informações necessárias para emissão, cobrança e baixa de boletos bancários.

Tipos de cobrança, dias para protesto e cálculo de juros, são apenas algumas dos dados contidos no arquivo. Esses arquivos ainda seguem o padrão CNAB estabelecido pela FEBRABAN (Federação dos Bancos) assim como outros órgãos tais como: ABAC, ANFAVEA, SINDIPEÇAS e etc.

Vale lembrar que os tais arquivos só funcionam se houver

uma boa documentação e se seguirem rigorosas regras de validação e checagem dos *bytes* envolvidos na transmissão. Para entendermos melhor como funcionam e onde se aplica essa tecnologia elaborei um exemplo de envio de notas fiscais a uma empresa de contabilidade. Mãos na massa!

Preparando o banco de dados de exemplo

Para este exemplo, criaremos um banco de dados utilizando o Firebird 1.5 e uma tabela de Notas Fiscais com alguns registros. A estrutura do banco encontra-se no *script* da **Listagem 1**.

Com o banco de dados criado vamos ao exemplo em si. Recomendo a abertura do Delphi 7, um novo projeto e a criação de uma janela semelhante à **Figura 1**.

O filtro utilizado para selecionar as notas fiscais a serem geradas, nesse caso, é baseado apenas na data de emissão das mesmas. Porém é bom ter em mente que cada caso requer uma atenção especial afim de selecionar as informações corretas e coerentes para popular o arquivo de EDI.

Antes de começamos a desenvolver a rotina para filtragem e geração do em si, precisamos preparar quatro *procedures* que auxiliarão na confecção correta das *strings* usadas no preenchimento do texto. São elas:

- *PadL*: usada para alinhar os textos à esquerda. Tem como função inserir caracteres em branco à direita da *string* até o limite estabelecido no segundo parâmetro;
- *PadR*: o contrário da *PadL*, ou seja, é usada para alinhar os textos à direita. Isso significa que a função insere caracteres em branco à esquerda da *string* até o limite estabelecido no segundo parâmetro;
- *LimparString*: será usada para retirar caracteres especiais da *string*, como: pontos, traços ou barras. Ideal para limpar máscaras de CNPJ, CPF etc.;
- *IncluirPonto*: em alguns campos de valor será necessário formatar incluindo pontos de decimal e milhar.

Para incluí-las no projeto declare-as na seção *public* ou *private* do formulário principal conforme a **Listagem 2** e pressione CTRL+SHIFT+C para que o Delphi crie o cabeçalho das mesmas.

O código completo da implementação das *procedures* pode ser verificado na **Listagem 3**.

Figura 1. Exemplo da layout do projeto

Listagem 1. Criação do banco de dados e tabela

```
SET NAMES WIN1251;

CREATE DATABASE 'localhost:<caminho>\EDI.FDB'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 4096
DEFAULT CHARACTER SET WIN1251;

CREATE TABLE NOTASFISCAIS (
  NUMERO            DOUBLE PRECISION NOT NULL,
  DOCUMENTO         VARCHAR(2),
  SERIE             VARCHAR(2),
  DATA_EMISSAO      DATE,
  UF_DESTINO        VARCHAR(2),
  NOME_RESPONSAVEL  VARCHAR(31),
  UF_RESPONSAVEL    VARCHAR(2),
  CGC_PAGADOR       VARCHAR(18),
  CGC_EMITENTE      VARCHAR(18),
  IE_RESPONSAVEL    VARCHAR(18),
  CFOP              VARCHAR(6),
  TOTAL_NOTA        NUMERIC(15,2),
  BASE_DE_CALCULO   NUMERIC(15,2),
  ALIQUOTA          NUMERIC(15,2),
  VALOR_IMPOSTO     NUMERIC(15,2),
  DIVERSOS          NUMERIC(15,2),
  BASE_IPI          NUMERIC(15,2),
  NAO_TRIBUTADO     NUMERIC(15,2),
  IPI               NUMERIC(15,2),
  OBS               VARCHAR(62)
);
ALTER TABLE NOTASFISCAIS
ADD CONSTRAINT PRIMARIO_PK
PRIMARY KEY (NUMERO);
```

Listagem 2. Declaração das funções auxiliares

```
function PadL(ATexto: string;
  ATamanho: Integer): string;
function PadR(ATexto: string;
  ATamanho: integer): string;
function LimparString(ATexto,
  ACaracteres: string): string;
function IncluirPonto(ATexto: string): string;
```

Listagem 3. Funções auxiliares

```
function TfrmPrincipal.PadR(ATexto: string;
  ATamanho: integer): string;
var
  i: integer;
begin
  Result := ATexto;
  for i := 1 to (ATamanho - Length(Result)) do
    Result := ' ' + Result;
end;

function TfrmPrincipal.PadL(ATexto: string;
  ATamanho: Integer): string;
var
  i: integer;
begin
  Result := ATexto;
  for i := 1 to (ATamanho - Length(Result)) do
    Result := Result + ' ';
end;

function TfrmPrincipal.LimparString(ATExto,
  ACaracteres: string): string;
var
  strAux: string;
  i: integer;
begin
  strAux := '';
  for i := 1 to Length(ATexto) do
    if Pos(Copy(ATexto, i, 1), ACaracteres) <= 0 then
      strAux := strAux + Copy(ATexto, i, 1);
  Result := strAux;
end;

function TfrmPrincipal.IncluirPonto(
  ATexto: string): string;
var
  i: integer;
begin
  Result := '';
  for i := 1 to Length(ATexto) do
    if ATexto[i] = ',' then
      Result := Result + '.'
    else
      Result := Result + ATexto[i];
end;
```

Nesse ponto temos condições de partir para a próxima parte do exemplo. Vamos criar o Data Module que conterá as conexões necessárias com o banco de dados e os componentes usados na filtragem e geração.

Crie um Data Module através do menu *File>New>Data Module*. Dê o nome de "dmPrincipal" e salve-o como "dmodPrincipal.pas". Adicione os componentes *SQLConnection* e *SQLDataSet* da paleta *dbExpress*.

Para o *SQLConnection* dê o nome de "SqlConexao" e conecte-o ao banco de dados *EDI.fdb*. Logo em seguida ligue o *SQLDataSet* ao *SQLConnection* através da propriedade *SqlConnection* e mude sua propriedade *Name* para "sdsNotasFiscais".

Ainda no *SQLDataSet* adicione no *CommandText* a seguinte instrução SQL:

```
SELECT * FROM NOTASFISCAIS
WHERE DATA_EMISSAO BETWEEN :pDATA_INI AND :pDATA_FIM
```

Adicione um *DataSetProvider*, um *ClientDataSet* e *DataSource* da paleta *DataAccess*. Nomeie-os como "dspNotasFiscais", "cdsNotasFiscais" e "dtsNotasFiscais" respectivamente. Faça a ligação entre os componentes (não esqueça de clicar com o botão direito no *cdsNotasFiscais* e escolher a opção *Fetch Params*). O Data Module deverá estar parecido a **Figura 2**.

Depois de prontas as configurações do Data Module, podemos retornar ao formulário principal e desenvolver a geração do arquivo em si.

Filtrando e escrevendo no arquivo texto

Em nosso exemplo vamos utilizar uma variável do tipo *TextFile* para iniciar/criar o arquivo e escrever nele tudo que precisamos de acordo com o *layout* estipulado. Em nosso *layout* de exemplo há três tabelas representando nome do campo, tamanho, posições de inicio e fim, valor quando fixo e formato.

As tabelas representam as especificações do *Header*, *Detail* e *Trailler* do arquivo respectivamente. O *Header* é a primeira linha do EDI, normalmente especifica o tipo de registro, nome da empresa, data de emissão e outras informações relevantes pra identificar que e de que empresa são os dados contidos no arquivo.

O *Detail*, ou detalhe, são os registros efetivamente que serão carregados no destino, nesse caso, Notas Fiscais para contabilidade. O *Trailler* é última linha do EDI e normalmente é usada para especificar quantidade de *Details* contidos no arquivo.

Outras considerações a respeito do *layout* são:

1. Todos os campos TEXTO são alinhados a esquerda;
2. Todos os campos NUMERICOS são alinhados à direita;
3. Campos reservados normalmente são de uso do destino;
4. Campos que contêm especificação de formato precisam ser registrados com a máscara determinada no *layout*.

Veja nas **Tabelas 1**, **2** e **3** a descrição do *layout* do arquivo.

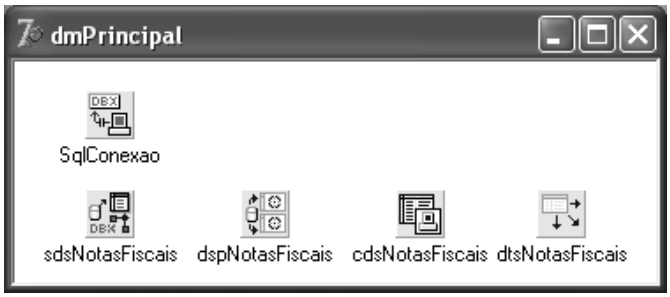


Figura 2. Data Module configurado para NotasFiscais

Feitas todas as considerações, basta escrever o código que vai filtrar, criar o arquivo em memória, escrever no mesmo e salvar. Então codifique o botão *Gerar* (evento *OnClick*) conforme a **Listagem 4**.

Entendendo o código

A programação do EDI não é segredo e portando nenhum bicho de sete cabeças. Vamos analisar o código pra que possamos entender melhor todo o processo de geração do arquivo *Contabilidade.txt*.

Logo de início declaramos uma variável (*Txt*) do tipo *TextFile*. Essa variável é usada em toda extensão do exemplo. É a representação em memória do arquivo *contabilidade.txt*. Iniciamos *Txt* com o comando *AssignFile* informando a variável de memória (primeiro parâmetro) e o endereço físico do arquivo (nesse caso, *C:\EDI\Contabilidade.txt*).

Em seguida conectamos ao banco, filtramos e preparamos o *cdsNotasFiscais* para a escrita no arquivo, conforme podemos ver o trecho do código anterior, na **Listagem 5**.

Note que há alguns tratamentos de erro, como por exemplo, a verificação se nas datas informadas existem dados para serem exportados. Nesse ponto já podemos começar a escrita do arquivo e pra iniciar devemos escrever as informações do *Header* conforme o *layout*, utilizando o código:

```
Write(Txt, '60.330.446/0001-01');
Write(Txt, '111.458.777.333');
Write(Txt, FormatDateTime('MM', Now));
WriteLn(Txt, PadL('REVISTA DEVMEDIA - CLUBEDELPHI', 40));
```

Após o término de escrita do *Header*, segue-se para a escrita do *Detail*, ou seja, as Notas Fiscais em si. Basta dar um *loop* no *ClientDataSet* e chamar a função *Write* para cada informação que deva ser enviada para o arquivo.

Note que os campos texto são enviados em conjunto da função *PadL* que faz a inclusão de caracteres espaço à direita da *string* pra completar o tamanho da mesma. Já os campos numéricos são utilizados, quando necessários, em conjunto com *PadR* incluindo espaços em branco à esquerda do número.

Em alguns *layouts* é comum utilizar-se de zeros à esquerda. Isso normalmente é estipulado pelo desenvolvedor do *layout*. Ao término do *loop* escrevemos o *Trailler* do arquivo e o fechamos (*CloseFile*) para que o arquivo seja efetivamente salvo.

Por último informamos ao usuário que o arquivo foi salvo com sucesso e o local onde se encontra o mesmo. Execute o projeto e veja o arquivo gerado (**Figura 3**).

CAMPO	TAMANHO/TIPO	INICIAL A FINAL	VALOR	FORMATO
CNPJ_EMITENTE	018 (TEXTO)	001 A 018		99.999.999/0009-99
IE_EMITENTE	015 (TEXTO)	019 A 033		999.999.999.999
MES REFERENCIA	002 (TEXTO)	034 A 035		MM
EMPRESA	040 (TEXTO)	036 A 076		

Tabela 1. Header do arquivo de layout

CAMPO	TAMANHO/TIPO	INICIAL A FINAL	VALOR	FORMATO
TIPO DOCUMENTO	004 (TEXTO)	001 A 004	NF,NFNS,CTRC	
SERIE	002 (TEXTO)	005 A 007	UM	
NUMERO NOTA	006 (NUMERO)	008 A 013		999999
DATA EMISSAO	005 (DATA)	014 A 018		DD/MM
UF_DESTINO	002 (TEXTO)	019 A 020		
NOME_RESPONSAVEL	030 (TEXTO)	021 A 050		
UF_RESPONSAVEL	002 (TEXTO)	051 A 052		
CGC_PAGADOR	016 (TEXTO)	053 A 067		99999999999999
IE_RESPONSAVEL	018 (TEXTO)	068 A 086		9999999999999999
CONTROLE	003 (TEXTO)	089 A 091	V21	FIXO
CFOP	004 (TEXTO)	092 A 098		999999
TOTAL_NOTA	014 (NUMERO)	099A 112		99999999999.99
BASE_DE_CALCULO	014 (NUMERO)	113 A 126		99999999999.99
ALIQUOTA	014 (NUMERO)	113 A 140		99999999999.99
RESERVADO	014 (NUMERO)	141 A 154		99999999999.99
VALOR IMPOSTO	014 (NUMERO)	155 A 168		99999999999.99
RESERVADO	014 (NUMERO)	169 A 182		99999999999.99
RESERVADO	014 (NUMERO)	183 A 196		99999999999.99
OBS	060 (TEXTO)	197 A 256		LIVRE
RESERVADO	002 (TEXTO)	257 A 258	00	FIXO
RESERVADO	001 (TEXTO)	259 A 296		BRANCOS
RESERVADO	002 (TEXTO)	297 A 298	01	FIXO

Tabela 2. Detail do arquivo de layout

CAMPO	TAMANHO/TIPO	INICIAL A FINAL	VALOR	FORMATO
DATA ARQUIVO	008 (DATA)	001 A 008		YYYYMMDD
RESERVADO	040 (TEXTO)	009 A 048		BRANCOS
REGISTROS	006 (NUMERO)	049 A 055		999999

Tabela 3. Trailler do arquivo de layout

Conclusão

Neste artigo vimos como desenvolver facilmente arquivos texto, seguindo padrões pré-definidos para integrar nossos registros de dados aos sistemas de outras empresas, tais como Bancos, Contabilidades etc.

É importante lembrar que não há fórmula mágica para a criação de EDI, ou seja, basta seguirmos o *layout* e regras passadas, que tudo será desenvolvimento mais rapidamente e sem ocorrências de erros. ■

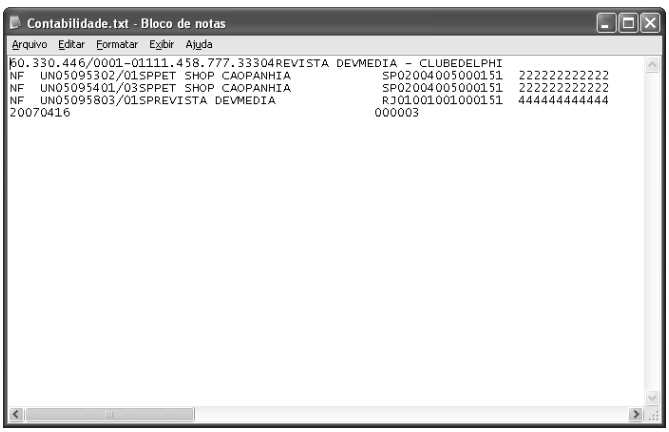


Figura 3. Contabilidade.txt gerado com sucesso

Listagem 5. Trecho onde filtramos e preparamos o cdsNotasFiscais

```
with dmPrincipal, sqlConexao, cdsNotasFiscais do
begin
  try
    if not Connected then
      Connected := True;
  except on E: Exception do
    ...
  end;

  { Passagem dos parâmetros para filtro }
  Params.ParamByName('pData_Ini').AsSQLTimeStamp :=
    DateTimeToSQLTimeStamp(dtpInicial.Date);
  Params.ParamByName('pData_Fim').AsSQLTimeStamp :=
    DateTimeToSQLTimeStamp(dtpFinal.Date);
  Active := True;
  if IsEmpty then
  begin
    ...
  end;
  ProgressBar1.Max := cdsNotasFiscais.RecordCount;
```

Listagem 4. Código do botão Gerar

```
uses
  SqlTimSt, uDM;
...
procedure TfrmPrincipal.BitBtn1Click(Sender: TObject);
var
  Txt: TextFile;
  vrlImposto: Real;
  NumRegistros: Integer;
begin
  ProgressBar1.Position := 0;
  ProgressBar1.Max := 0;
  { Criação do arquivo de texto }
  ForceDirectories('C:\EDI\');
  AssignFile(Txt, 'C:\EDI\Contabilidade.txt');
  ReWrite(Txt);
  with dmPrincipal, sqlConexao, cdsNotasFiscais do
  begin
    try
      if not Connected then
        Connected := True;
    except on E: Exception do
      begin
        MessageDlg('Não foi possível estabelecer '+
          'conexão com o banco de dados.' + #13 +
          'Erro original:' + #13 + E.Message, mtError,
          [mbOk], 0);
        Halt;
      end;
    end;
    { Passagem dos parâmetros para filtro}
    Params.ParamByName('pData_Ini').AsSQLTimeStamp :=
      DateTimeToSQLTimeStamp(dtpInicial.Date);
    Params.ParamByName('pData_Fim').AsSQLTimeStamp :=
      DateTimeToSQLTimeStamp(dtpFinal.Date);
    Active := True;
    if IsEmpty then
    begin
      MessageDlg('Não foram encontradas Notas Fiscais'+
        ' nesse período!', mtInformation, [mbOk], 0);
      dtpInicial.SetFocus;
      Exit;
    end;
    ProgressBar1.Max := cdsNotasFiscais.RecordCount;
    { Header do arquivo }
    Write(Txt, '60.330.446/0001-01');
    Write(Txt, '111.458.777.333');
    Write(Txt, FormatDateTime('MM', Now));
    WriteLn(Txt, PadL('REVISTA DEVMEDIA - CLUBEDELPHI', 40));
    NumRegistros := 0;
    { Detail do Arquivo }
    while not cdsNotasFiscais.EOF do
    begin
      Write(Txt, 'NF  ');
      Write(Txt, 'UN');
      Write(Txt, PadR(FormatFloat('000000',
        FieldByName('NUMERO').AsInteger), 6));
      Write(Txt, FormatDateTime('DD/MM',
        SqlTimeStampToDatetime(FieldByName(
        'DATA_EMISSAO').AsSQLTimeStamp)));
      Write(Txt, FieldByName('UF_DESTINO').AsString);
      Write(Txt, PadL(FieldByName(
        'NOME_RESPONSAVEL').AsString, 30));
      Write(Txt, PadL(FieldByName(
        'UF_RESPONSAVEL').AsString, 2));
      Write(Txt, PadL(LimparString(FieldByName(
        'CGC_PAGADOR').AsString, './-'), 16));
      Write(Txt, PadL(LimparString(FieldByName(
        'IE_RESPONSAVEL').AsString, './-'), 18));
      Write(Txt, 'V12');
      Write(Txt, PadL(FieldByName('CFOP').AsString, 6));
      Write(Txt, PadR(IncluirPonto(FormatFloat(
        '00000000000.00', FieldbyName(
        'TOTAL_NOTA').AsFloat)), 14));
      Write(Txt, PadR(IncluirPonto(FormatFloat(
        '00000000000.00', FieldByName(
        'BASE_DE_CALCULO').AsFloat)), 14));
      Write(Txt, PadR(IncluirPonto(FormatFloat(
        '00000000000.00', FieldByName(
        'ALIQUOTA').AsFloat)), 14));
      Write(Txt, PadR(IncluirPonto(FormatFloat(
        '00000000000.00', 0)), 14));
      { Calcula o valor do Imposto }
      vrlImposto := Round(FieldbyName(
        'TOTAL_NOTA').AsFloat - FieldByName(
        'BASE_DE_CALCULO').AsFloat);
      Write(Txt, PadR(IncluirPonto(FormatFloat(
        '00000000000.00', vrlImposto)), 14));
      Write(Txt, PadR(IncluirPonto(FormatFloat(
        '00000000000.00', 0)), 14));
      Write(Txt, PadR(IncluirPonto(FormatFloat(
        '00000000000.00', 0)), 14));
      Write(Txt, PadL(FieldByName('OBS').AsString, 60));
      Write(Txt, '00');
      Write(Txt, PadL(' ', 40));
      WriteLn(Txt, PadL('01', 2));
      ProgressBar1.Position := ProgressBar1.Position + 1;
      cdsNotasFiscais.Next;
      Inc(NumRegistros);
    end;
    { Trailler do Arquivo }
    Write(Txt, FormatDateTime('YYYYMMDD', Now));
    Write(Txt, PadL(' ', 40));
    Write(Txt, PadR(FormatFloat('000000', NumRegistros), 6));
  end;
  CloseFile(Txt);
  MessageDlg('Arquivo de EDI para contabilidade '+
    'gerado com sucesso!' + #13 +
    'Gerado o arquivo [C:\EDI\Contabilidade.txt].',
    mtInformation, [mbOk], 0);
end;
```

Seu potencial. Nossa inspiração.™
Microsoft
ANTES DE ENCARAR UMA BATALHA
NA INTERNET, ESCOLHA A ARMA CERTA.
ENFRENTE OS DESAFIOS
Com o Visual Studio®, você tem os controles ASP.NET e AJAX para comandar qualquer browser e ainda aumentar suas habilidades para construir sites mais interativos. Veja todas as dicas e informações em www.enfrenteosdesafios.com.br
© 2007 Microsoft Corporation. Todos os direitos reservados. Microsoft, Microsoft® Visual Studio® 2005 e "Seu potencial. Nossa inspiração." são marcas, registradas ou não, da Microsoft Corporation nos Estados Unidos e/ou em outros países.
Microsoft
Visual Studio

Vem aí
Web
Mobile
TechWeek
A 2ª Edição do maior
evento de desenvolvimento
Web e Wireless
do país.
10 e 11 de Agosto São Paulo / SP
44 sessões, 100% técnicas apresentando o que as tecnologias
Java e .NET trazem de melhor para este mercado.
Prepare-se e não fique de fora!
Economize e compre seu ingresso antecipado:
Para mais informações:
www.devmedia.com.br/eventos/webmobiletechweek
DevMedia
GROUP